VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Detekce pohybu ve videosekvencích

Motion Detection in Surveillance Video

2016

Michal Šimon

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Michal Šimon**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Detekce pohybu ve videosekvencích
Motion Detection in Surveillance Video

Jazyk vypracování: čeština

Zásady pro vypracování:

Detekce pohybu ve videosekvencích je jednou z důležitých operací ve zpracování obrazu. Uplatnění nalézá v mnoha aplikacích, např. automatické střežení, registrace osob a věcí apod. Cílem této diplomové práce je popsat aktuální algoritmy užívané pro detekci pohybu, jeden z nich prakticky implementovat a řádně otestovat.

1. Seznamte se současnými metodami užívanými pro detekci pohybu.
2. Proveďte rešerši současných algoritmů publikovaných na odborných konferencích a v monografiích. Pokuste se tyto algoritmy porovnat.
3. Vyberte některou ze současných metod (publikovanou po roce 2010) a tu implementujte.
4. Výslednou implementaci otestujte a porovnejte s výsledky jiných metod (např. pomocí standardizovaných datasetů).
5. V závěru zhodnoťte své výsledky a popište možnosti dalšího vývoje.

Pro implementaci je možno využít knihovnu OpenCV.

Seznam doporučené odborné literatury:

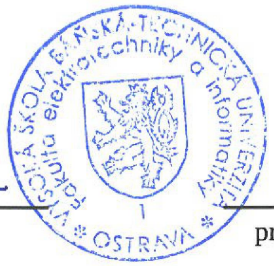
[1] Gary Bradski, Learning OpenCV: Computer Vision with the OpenCV Library, O'Reilly Media, 2008, ISBN 0596516134
[2] Amar Mitiche, J.K. Aggarwal, Computer Vision Analysis of Image Motion by Variational Methods (Springer Topics in Signal Processing), Springer, 2014, ISBN 3319007106
[3] Panca Mudjirahardjo et al., Abnormal Motion Detection in an Occlusive Environment, Proc. SICE 2013, 2013

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Mgr. Ing. Michal Krumnikl, Ph.D.**

Datum zadání: 01.09.2014

Datum odevzdání: 29.04.2016



doc. Dr. Ing. Eduard Sojka	prof. RNDr. Václav Snášel, CSc.
vedoucí katedry	*děkan fakulty*

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 27. dubna 2016

Michal Šimon

.....

Souhlasím se zveřejněním této diplomové práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v magisterských programech VŠB-TU Ostrava.

V Ostravě 27. dubna 2016

Michal Šimon

.....

Rád bych na tomto místě poděkoval svému vedoucímu Mgr. Ing. Michal Krumnikl, Ph.D. za jeho ochotu, trpělivost a cenné rady. Dále bych rád poděkoval své rodině a přátelům za morální podporu a v neposlední řadě také své kamarádce Květoslavě Liškové za pomoc s jazykovou korekturou práce.

Abstrakt

Tato práce popisuje základní metody detekce pohybu ve videosekvencích a rozebírá rozdílné přístupy k řešení této úlohy. Jsou zde popsány výhody a nevýhody jednotlivých přístupů a vhodnost jejich použití na konkrétních příkladech. Dále je rozebrán článek Abnormal Motion Detection in an Occlusive Environment jako základ pro implementaci a rozšíření této metody. Součástí této práce je také implementace analyzátoru jako cloudové služby pomocí NodeJS a nativního modulu v jazyce C++. Toto spojení zajišťuje snadnou škálovatelnost a zároveň dostatečný výpočetní výkon. Je zde rozebrán také vhodný přístup k vývoji takovéto služby a popsán proces automatizace sestavení aplikace i konfigurace infrastruktury. Práce obsahuje také testy analyzátoru na několika existujících datasetech včetně výroby vlastního.

Klíčová slova: Optický tok, Oddělení popředí a pozadí, DBScan, K-means, Interoperabilita C++ a NodeJS, Cloud, Automatizace

Abstract

This paper describes the basic methods of motion detection in video sequences and discusses different approaches to solving this task. There are described advantages and disadvantages of different approaches and their suitability to specific examples. Furthermore there is described the article Abnormal Motion Detection in an occlusive Environment as a basis for the implementation and expansion of this method. Part of this work is the implementation of the analyzer as a cloud based service using NodeJS and native module in C++. This connection provides easy scalability and also enough computing power. There is also analyzed the appropriate approach to the development of such services and described the process of application assembly automation and configuration of infrastructure. This work also includes test for analyzer on several existing datasets, including the production of own one.

Key Words: Optical Flow, Foreground and background separation, DBScan, K-means, C++ and NodeJS interoperability, Cloud, Atomation

Obsah

Seznam použitých zkratek a symbolů

AWS	–	Amazon Web Services
EC2	–	Elastic Compute Cloud
SSL	–	Secure Socket Layer
SaaS	–	Software as a Service
HTML	–	Hypertext Markup Language
CSS	–	Cascading Style Sheets
JS	–	JavaScript
ES2015	–	ECMAScript 2015
IoT	–	Internet of Thinks
IOPS	–	Input/Output operations Per Second
YAML	–	YAML Ain't Markup Language
NFS	–	Network File System
VPS	–	Virtual Private Server

Seznam obrázků

Seznam tabulek

1 Úvod

Cílem následujících kapitol je vysvětlit základní pojmy a principy obecné detekce pohybu objektů ve videosekvencích. Dále se budu zabývat samotnou detekcí pohybu a srovnáním různých metod a algoritmů, z nichž jeden 3 jsem se rozhodl implementovat a otestovat. Pokusil jsem se zvolený algoritmus i rozšířit a vylepšit jeho výsledky.

V další části práce 4 demonstruji také základní principy správného přístupu k vývoji, udržování a nasazení aplikace do produkčního prostředí. Popisuji nástroje a postupy pro správný návrh softwaru, jako služby (SaaS), a navrhuji také další možnosti rozšíření a optimalizace nejen samotné detekce, ale také technické části. Výstupem mé práce je samotná implementace analyzátoru, který důkladně rozebírám na dílčí části, popisuji jejich vnitřní i vnější vlastnosti a také interoperabilitu mezi sebou. V této problematice se zabývám automatizací nejen vývojového procesu, ale také samotné infrastruktury pro provoz takovéto cloudové aplikace.

V části Experimenty 5 se zabývám kvalitou samotné detekce a její měřitelností. Popisuji také problémy, na které jsem narazil spolu s výsledky měření a konkrétní konfigurace, za které jsem jich dosáhnul, včetně ukázek vizualizace výsledků analyzátoru. Vše doplňuji o vlastní data-set a popisuji jeho přípravu a výsledky jeho analýzy.

Věřím, že poskytuji zajímavý náhled do problematiky analýzy obrazu a přispívám v tomto směru netradičním přístupem k implementaci a praktické aplikaci.

1.1 Analýza obrazu

Analýza obrazu je v posledních několika letech skloňovaná nejen na akademické půdě, ale čím dál častěji také v komerční sféře. Díky rostoucímu výkonu výpočetní techniky jsou nejrůznější algoritmy pro aplikaci počítačového vidění mnohem dostupnější a spolehlivější. Dnes se tedy můžeme setkat s velkým množstvím reálných využití této disciplíny v každodenním životě, například v automobilovém průmyslu, dopravě nebo při zabezpečení veřejných prostor.

Ve své bakalářské práci jsem se již této problematice částečně věnoval a zaujala mne natolik, že jsem se rozhodl volně v této činnosti pokračovat i ve své diplomové práci. Zabýval jsem se problémy kategorizace a vyhledávání statických fotografií. Nyní bych chtěl své poznatky posunout dále a pokusit se proniknout hlouběji do problematiky videosekvencí a jejich analýzy.

1.2 Počítačové vidění

Disciplína detekce pohybu spadá do široké oblasti počítačového vidění, která si klade za cíl, pomocí softwaru správně detekovat objekty v obraze a porozumět jim podobně jako lidé. Tato disciplína je ovšem velmi nesnadná pro počítač a úspěšnost konkrétní aplikace často závisí na mnoha faktorech, jako jsou například světelné podmínky scény, kvalita pořízeného záznamu nebo vzájemná poloha snímače a objektů.

Vidění, tak jak jej dokáže člověk, je oproti počítači v mnoha směrech ve výhodě. Jeho fyzikální vlastnosti jsou totiž na mnohem lepší úrovni, než často dokážeme obstarat pro počítač ze strany pořízeného záběru scény. Jedním z nejpodstatnějších rozdílů je, že člověk vidí své okolí hned pomocí dvou snímačů (očí), které mají vůči sobě neměnnou polohu v hlavě. Tento fakt umožňuje lidskému mozku vnímat prostor díky stereoskopii ve třech dimenzích, oproti počítači, který často pracuje pouze s dvojrozměrným obrazem. Samotný převod 3D prostoru do 2D zobrazení s sebou přináší mnoho komplikací, které vedou vždy ke ztrátě důležitých informací o scéně.

Dalším důležitým rozdílem je lidská paměť. Ta umožňuje člověku se postupně od narození učit různé vzory a průběžně trénovat jejich rozpoznávání. Tyto jednoduché vzory následně skládat do složitějších vzorů, které nám pak umožní mnohem snadněji a rychleji rozpoznat objekt v reálném světě (3D) či na obrázku (2D). Tyto vzory jsou základní geometrické obrazce, ze kterých skládáme složitější objekty. Například složitější vzor jízdního kola si sestavíme z jednodušších vzorů, tedy dvou kruhů a několika přímek či křivek. Takovýchto obecných i specifických vzorů si za dobu svého života člověk vytvoří obrovskou sadu s přiřazenými významy ve své paměti. Při vizuální detekci objektu pak ve velkém množství případů tuto databázi využije ke zrychlení a zkvalitnění kategorizace předmětu nebo objektu. Naopak, pokud se člověk setká s předmětem, který doposud nespatřil, jeho detekce mu trvá podstatně déle a snaží se využít další dostupné způsoby k získání maxima informací o objektu, jako například prohlédnutí si předmětu z více úhlů či zahrnutí dalších smyslových vjemů. Například pomocí čichu, teploty nebo dalších zkušeností a vlastností jako je hmotnost, pružnost a podobně, dokáže urychlit proces rozpoznání.

Většinu těchto podstatných informací bohužel nejsme schopni počítači předat, natož s předmětem dle aktuálních analyzovaných poznatků interaktivně manipulovat. Je tedy logické, že počítačové vidění má oproti lidskému mozku mnohem těžší podmínky a nemůžeme očekávat s tak omezeným množstvím informací srovnatelné výsledky s lidským viděním. Přesto některé specifické implementace počítačového vidění dosahují překvapivě dobrých výsledků, pokud jsou aplikovány na úzký okruh problému. Klíčem ke kvalitní implementaci je tedy úzké vymezení této disciplíny pro konkrétní aplikaci a její následná optimalizace na omezené množině objektů.

1.3 Detekce pohybu

Pohyb jako takový je z fyzikálního hlediska dosti rozsáhlé téma, a proto se jej pokusím pro účely této práce lehce konkretizovat. Základními prvky ve scéně jsou pro nás vždy sledované objekty, prostředí, ve kterém se vyskytují a také kamera, kterou provádíme projekci 3D prostoru do dvojrozměrných snímků v sekvenci. Je nejspíše zřejmé, že všechny tyto součásti pozorovaného prostoru mohou být v pohybu, ale i v klidu vůči sobě navzájem. Pokud se tedy bude pohybovat kamera společně s objektem po stejné trajektorii, bude v pořízené videosekvenci tento objekt chybně chápán jako statický (v klidu), ale veškeré okolí se bude zdát v pohybu. V opačném případě předpokládejme, že je kamera vůči prostředí v klidu, ale je v pohybu pouze objekt. V takové videosekvenci pak budeme pohyb chápat správně v souladu s realitou. Je tedy potřeba

v takovýchto situacích správně počítači tyto parametry sdělit nebo je alespoň zohlednit při čtení analyzovaných dat.

Samotný pohyb lze ve 2D obraze detekovat několika rozdílnými přístupy. Mezi nejčastěji používané principy patří oddělení pozadí a popředí a detekce optického toku.

2 Pojmy

Ještě předtím, než pronikneme do detailů této problematiky, bylo by vhodné si ujasnit nejprve několik důležitých pojmů. Pokusím se proto alespoň stručně popsat některé klíčové metody z oblasti analýzy obrazu a přiblížit čtenáři jejch základní vlastnosti. Zmíním také vhodné použití metod a jejich výhody a nevýhody v konkrétních situacích.

2.1 Oddělení popředí a pozadí

Tato technika je často také nazývána jako oddělení pozadí či detekce popředí. Jak sám název napovídá, princip je založen na modelování pozadí jako určité masky, pomocí které se pak odečtením od původního snímku vyextrahuje popředí, tedy detekované objekty. Často je doplněn ještě třetí krok validace dat, který má za úkol ověřit správnost detekovaných objektů. Při odečtení se totiž neberou v potaz sousední pixely, a proto získáme pouze hrubý odhad popředí, což může vést k náhodnému šumu v masce. Toho se můžeme zbavit pomocí morfologických filtrů [13], které dokáží odstranit osamocené pixely (také je nazýváme sirotky).

2.1.1 Modelování pozadí

Modely pozadí mohou být buď statické nebo dynamické. Jejich odlišnost je především v tom, jak se dokáží přizpůsobovat změnám v obraze. Statické modely se spočítají dopředu a po dobu celé videosekvence se nemění. To má sice velký benefit v oblasti výkonu, ale komplikuje to zpracování například streamovaného videa či správnou reakci na změnu světelnosti ve scéně.

Dynamické modely umožňují naopak adaptivně v čase upravovat masku tak, aby se neobjevovali v popředí "duchové". Tento jev může vzniknout například náhlým zrychlením projíždějícího vozidla, přičemž model se nezmění dostatečně rychle na to, aby dokázala masku upravit natolik, aby se s tímto chováním objektu dokázala vypořádat v reálném čase. Opačný problém může naopak nastat v situaci, kdy se například chodec na chvíli zastaví, čímž splyne s pozadím a je tedy detekován chybně.

K modelování pozadí lze přistupovat ještě dle dalšího kritéria. Jedná se o množství snímků, které se zohledňují při výpočtu modelu. Tyto techniky můžeme rozdělit na rekurzivní a nerekurzivní. První skupina zohledňuje ve výpočtu rekurzivně vždy všechny předchozí snímky. Výhodou tohoto přístupu je, že i pomalu se pohybující objekty či úplné zastávky v jejich trajektorii jsou detekovány správně, protože vycházíme z kompletní historie jejich chování. Naopak tento přístup dokáže ovlivnit i jediný chybný snímek, který bude mít dopad na všechny následující. Také se špatně přizpůsobuje změnám v prostředí, adaptace změny trvá dlouho. Mezi tyto metody patří například: Přibližný mediánový filtr (Approximated Median Filter), Mixtura Gausiánů nebo Kalmanův filtr. Podrobněji jsou tyto metody popsány v publikacích [6], [7] a [8].

Nerekurzivní přístup si uchovává pouze sadu několika posledních snímků, ze kterých pak počítá model. Díky tomu se dokáže lépe vypořádat se změnami prostředí, ale má naopak pro-

blémy s delšími zastávkami objektů. Ty po určité době (délka sady) začne považovat za pozadí, protože již v sadě chybí informace o pohybu objektu. Mezi tyto metody patří například: Lineární prediktivní filtr, Rozdíl snímků (Frame differencing) nebo Mediánový filtr.

2.1.2 Výpočet pomocí rozdílu snímků

Pro snadné pochopení jsem si jako příklad vybral metodu rozdílu snímků, která patří mezi nerekurzivní metody a je velmi jednoduchá. Používá totiž k výpočtu 1 modelu pozadí pouze dva snímky. Aktuálně zpracovávaný snímek v čase t a předchozí v čase $t-1$. Pomocí rozdílu intenzit I v každém bodě obrazu

$$|I_t(x,y) - I_{t-1}(x,y)| > T \tag{1}$$

získáme informaci o velikosti změny mezi snímky. Tuto hodnotu porovnáme s určitým prahem citlivosti T, který nám pomůže oddělit nevýznamné změny od významných. Tato metoda je sice velmi jednoduchá, ale prakticky nepoužitelná, pokud objekty mají ve své trajektorii nějaké zastávky. Velikost sady předchozích snímku je zde totiž rovna jedné, což znamená, že okamžitě po zastavení objektu je přesunut z popředí do pozadí. Naopak adaptace na změny v prostředí je zde téměř okamžitá, což může být v některých případech velkou výhodou.

2.1.3 Detekce popředí

Model pozadí si můžeme představit opět jako obrázek. Od aktuálního snímku I_t pak stačí odečíst model pozadí B_t pomocí výpočtu 2. Výslednou hodnotu pro každý pixel pak porovnáme opět s prahem citlivosti T. Pokud je tedy práh citlivosti překonán, považujeme pixel (x, y) za popředí.

$$|I_t(x,y) - B_t(x,y)| > T \tag{2}$$

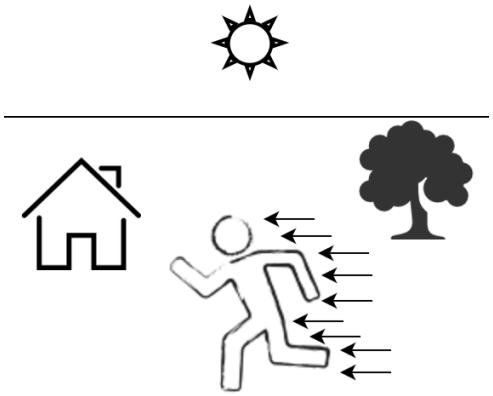
V případě, že potřebujeme popředí detekovat ve scéně, kde se vyskytují časové úseky s nižším kontrastem, je vhodnější použít například normalizovanou statistiku, kde se práh citlivosti počítá dynamicky s ohledem na aktuální situaci ve scéně. Jiné metody používají například prahy citlivosti dva pro odlišný přístup k málo kontrastním záběrům oproti standardním. Podrobnosti o metodách oddělení popředí a pozadí lze nalézt v publikacích [10] a [11].

2.2 Optický tok

Optický tok (Optical flow) je reprezentací pohybu v sekvenci snímků. Je vyjádřen polem vektorů, z nichž každý udává rychlost a směr pohybu sledovaného bodu mezi dvojicí snímků. Je nutné si uvědomit, že vektor rychlosti nemusí vždy reprezentovat skutečnou rychlost pohybu objektu kvůli následujícím problémům apertury (aperture problem):

- Pohyb kamery - v případě, že je snímač obrazu také v pohybu, je optický tok detekovaný v obraze pouze relativní k pohybu kamery, což může výrazně zkreslit výsledky detekce.

- Středová projekce - převod trojrozměrné scény do dvourozměrného obrazu může metodě činit problém pohyb detekovat vždy v jednom z rozměrů (dle orientace kamery). V případě, že se objekt pohybuje směrem hloubkového vektoru kamery, je projektován do sekvence obrazů pouze změnou velikosti, což bude mylně detekováno jako nepatrný pohyb.
- Nedostatek anomálií v obraze - optický tok lze vyjádřit jako posun intenzity v čase. Takto lze na problém nahlížet pouze za předpokladu, že máme v obraze dostatek zajímavých bodů (anomálií). Tyto body můžeme popsat jako místa, jejichž gradient intenzity není nulový. Názorným příkladem tohoto problému může být třeba pohyb objektu modré barvy po pozadí stejné barvy. V případě správného osvětlení s minimálním stínem může být pohyb takového tělesa nezachytitelný jak pomocí metody optického toku, tak i lidským okem.



Obrázek 1: Vizualizace optického toku

2.2.1 Výpočet optického toku

Optický tok můžeme chápat jako funkci intenzity 3, která se v čase mění. Mějme tedy obraz reprezentován jako $o = (x, y)$ a t jako dimenzi času. Samotnou funkci pak můžeme vyjádřit jako $I(o, t)$. Musíme však zajistit splnění předpokladu o zachování intenzity (intensity conservation assumption). Mějme tedy vektor rychlosti a směru v bodě o: $s = (u, v)^t$ a pomocí něj můžeme vyjádřit předpoklad jako:

$$I(o, t) = I(o - s\delta t, t - \delta t) \tag{3}$$

Jinými slovy lze říci, že intenzita obrazového bodu o v čase t je stejná, jako intenzita této funkce v čase $t - \delta t$ v bodě, jehož změna je závislá na velikosti optického toku. Tedy bod z předchozího snímku je promítnut do aktuálního snímku sekvence posunutý o vektor optického toku.

Častěji se můžeme setkat s vyjádřením rovnice optického toku jako:

$$I_x u + I_y v + I_t = 0 \tag{4}$$

kde u a v jsou horizontální a vertikální složka výsledného vektoru.
K samotnému výpočtu lze přistupovat hned několika metodami:

- Diferenční přístupy
- Vyhledávání oblastí
- Metody založené na energii
- Metody založené na fázi

Blíže se podíváme na první kategorii metod, protože jsou mnohem dostupnější z pohledu výkonu než ostatní. Podrobně jsou tyto metody popsány v publikaci *Performance of optical flow techniques* [2].

2.2.2 Diferenční přístupy

Tyto metody využívají parciálních derivací prvního či vyšších řádů, z nichž sestavíme diferenciální rovnici. Tato rovnice je pak určitou aproximací předpokladu zachování intenzity. Vzhledem k tomu, že tento předpoklad pouze aproximujeme, dochází k chybě, kterou se dále snažíme minimalizovat. Z důvodu problému s aperturou zavádíme ještě jeden chybový parametr, který pomáhá lépe určovat optický tok na místech, kde se gradient blíží k nule. Pro dosažení ještě většího zpřesnění lze zavést další omezující předpoklady. Díky nim je však výpočet složitější (vyšší řády diferenciální rovnice) a tyto metody jsou více náchylné na nedostatky obrazu. Například se jedná o šum či pohyb sledovaných objektů mimo záběr snímače.

2.2.3 Horn-Schunck

Tato metoda vychází z předpokladu o zachování intenzity, což odpovídá první rovnici optického toku. Další rovnici lze odvodit z předpokladu, že sousední obrazové body se příliš neliší svou rychlostí pohybu. Z tohoto důvodu je metoda vhodná spíše pro detekci plynulého pohybu ve scéně. Metoda proto preferuje neostré hrany. Prakticky funguje metoda na principu definování jednotlivých chybových parametrů z omezujících předpokladů a jejich následné minimalizaci. Tu provádíme součtem čtverců jednotlivých chybových členů.

Výpočet není vhodné provádět běžnými eliminačními metodami (například Gaus-Jordanova) kvůli složitosti, ale je nutné zvolit iteračními metodu (Jacobiho nebo Gauss–Seidelovy), což vyžaduje nemalý výpočetní výkon.

S iteračními metodami je spojen ještě problém určení správného počtu iterací. Ten lze stanovit například zjišťováním, zda relativní chyba neklesla pod určitou mez v každé iteraci, což

přináší další nároky na výpočetní výkon. Alternativou může být odhad počtu iterací z předchozího snímku za předpokladu, že směr a rychlost pohybu se mezi snímky výrazně nezmění (například opačný směr nebo velké zrychlení). Náročnost výpočtu této metody tak činí jednu z největších nevýhod. Více informací se lze o této metodě dozvědět v publikaci Determining Optical Flow [14].

2.2.4 Lucas-Kanade

Metoda Lucas-Kanade [15] je postavena na stejných základech jako Horn-Schunck. Novým principem, oproti výše zmíněné metodě, je rozdělení obrazu do menších sekcí Ω, nad kterými se vyhodnocují předpoklady zvlášť. Velikost sekce, respektive okolí bodu (x, y) je definováno pomocí W a uvnitř závorek již počítáme klasickou rovnici optického toku.

$$\sum_{x \epsilon \Omega} W^2 [I_x u + I_y v + I_t]^2 \tag{5}$$

Na rozdíl od metody Horn-Schunck netrpí L-K některými problémy, jako například vysokými nároky na výpočetní výkon a minimalizací chyby na globální úrovni. Iterační přístup vyžaduje výpočet konvoluce a optického toku pro každou dílčí iteraci. Naopak metoda LK aplikuje konvoluci pouze na zmíněné oblasti W, což je výrazně rychlejší.

Globální minimalizace chyby zde také není nutná. Zavádíme proto chybový člen pro každý bod obrazu. Váha chyby bodů okolí pak se vzdáleností slábne, což zajišťuje distribuci chyby do míst, kde se gradient blíží k nule.

$$\rho LK = \sum_{x \epsilon \Omega} W^2(x) [\Delta I(x, t) * v + I_t(x, t)]^2 \tag{6}$$

$W(x)$ jsou pak váhy jednotlivých bodů v okolí Ω. Drobnou nevýhodou tohoto způsobu distribuce je, že se pixely v daném okolí Ω vzájemně ovlivňují a proto mají podobnou rychlost optického toku. Teoreticky bychom mohli pomocí druhých parciálních derivací určovat zrychlení, čímž bychom mohli získat informaci o určité plynulosti pohybu.

2.2.5 Moravcův detektor

Tato metoda využívá malého okénka, které posouvá po obraze vertikálně i horizontálně a měří změnu průměrné intenzity. Tato metoda

$$E(x, y) = \sum_{u,v} [I(x + u, y + v) - I(x, y)]^2 \tag{7}$$

$I(x, y)$ zde vyjadřuje průměrnou hodnotu intenzity okénka před posunutím a $I(x + u, y + v)$ po posunutí. Posun u a v vždy patří množině $\{-1, 0, 1\}$. $E(u, v)$ je matice, obsahující kvadráty rozdílů intenzit obou okének. Tato metoda je účinná především v násobcích 45°, což je považováno za největší omezení.

2.2.6 Harris corner detector

Tato metoda [16] vznikla v roce 1988 zdokonalením Moravcova detektoru a jejím základním principem je také detekce rohů. Za rohy považujeme taková místa, kde se mění gradient ve dvou směrech najednou. Na rozdíl od Moravce je však tato metoda invariantní vůči úhlu. Je však výpočetně mnohem náročnější a také citlivější na šum. Řešením může proto být předřazený low-pass filter například pomocí konvoluce Gausiánu. Nesmí však dojít k přílišnému rozmazání obrazu, které by mohlo způsobit až absenci skutečně důležitých bodů zájmu.

2.3 Shlukování

Shlukování (clustering) je proces, při kterém se snažíme rozdělit skupinu prvků do dílčích podskupin na základě konrétních kritérií. Tato kritéria jsou většinou určena vlastnostmi těchto prvků. Díky tomu získáme v každé skupině prvky, které jsou si v určitém ohledu podobné.

2.3.1 DBScan

Celým názvem "Density-based spatial clustering of applications with noise"je označována tato metoda shlukování. Podobně jako například známá K-means nebo K-median má za úkol rozčlenit určité prvky do několika shluků. Tyto shluky jsou pak reperezentovány body s určitou společnou vlastností. Vstupních kritérií pro zařazení přitom může být $1 - n$.

Zásadní odlišností této metody oproti dalším zmíněným je fakt, že dopředu nepotřebujeme znát počet skupin, do kterých chceme prvky rozčlenit. Tento parametr bude pro naše použití později velmi zásadní. Hlavním principem této metody je určování vzdálenosti jednotlivých prvků od sebe. Na jednoduchém jednorozměrném příkladu si pojďme tento princip vysvětlit.

Na vstupu mějme konečnou množinu prvků M, tedy například pětici. Každý z těchto prvků je ohodnocen jediným parametrem, na základě kterého budeme dále prvky rozdělovat. Dalšími vstupními parametry jsou minimální počet prvků n ve skupině a také maximální vzdálenost dvojice prvků od sebe uvnitř skupiny m.

$$M = \{2, 3, 9, 11, 20\}; n = 2; m = 3 \tag{8}$$

Postup při řešení takovéto úlohy bude následující. Prvním krokem v závislosti na implementaci může být buď výběr prvního prvku nebo náhodný výběr libovolného startovního prvku množiny M. Začněme tedy od začátku výběrem prvního prvku s hodnotou 2. Ten označíme jako navštívený a v jeho nejbližším okolí hledejme sousední body do maximální vzdálenosti m. Tomuto kritériu odpovídá pouze prvek s hodnotou 3. Protože jsme již splnili druhé pravidlo o minimálním počtu prvků ve skupině, můžeme tyto dva prvky označit za skupinu A a přesunout se do prvku s hodnotou 3. Ten opět označíme jako navštívený a opakujeme hledání dalších bodů v okolí. Tentokrát jsme však žádné další body nenalezli.

Opět tedy začneme od začátku přesunem na další, doposud nenavštívený, bod pomocí pořadí či náhodného výběru. V našem případě se jedná o prvek s číslem 9, který má v sousedství také bod 11. Tyto prvky můžeme opět označit za skupinu B. Následně se opět přesuneme do posledního nenavštíveného prvku. Protože však nemá ve svém okolí žádného nenavštíveného souseda a ani nebyl přiřazen do skupiny obsahující alespoň 2 prvky, označíme tento prvek za šum. Výsledek clusteringu DBScan je tedy:

$$A = \{2, 3\}; B = \{9, 11\} \tag{9}$$

Tuto metodu lze snadno aplikovat na více dimenzionální prostory. Je však nutné, abychom dovedli určit vzdálenost jednotlivých prvků množiny od sebe pomocí jejich vlastností. Pro 2D prostor můžeme použít například Euklidovskou vzdálenost a tak dále.

3 Abnormal Motion Detection in an Occlusive Environment

Při hledání podkladů pro svou práci jsem narazil na zajímavý příspěvek z konference SICE Annual Conference od autorů Panca Mudjirahardjo, Joo Kooi Tan, Hyoungseop Kim a Seiji Ishikawa z japonské Nagoya University. Tento článek Abnormal Motion Detection in an Occlusive Environment [1] mne natolik zaujal, že jsem se rozhodl jej zvolit jako hlavní podporu své práce. V následujícím textu si vysvětlíme základní principy a algoritmy popsané v dokumentu a detailněji si rozebereme jejich nejdůležitější části.

Cílem této vědecké práce bylo sestavení algoritmu pro detekci abnormality pohybu. Základem je především vektor optického toku nad každým "zajímavým"bodem (feature point) obrazu. Díky informacím o směru a rychlosti pohybu bodů mezi jednotlivými snímky sekvence jsme schopni detekovat trajektorii a dynamiku pohybu bodů v čase. Následně z informací o délkách vektorů můžeme sestavit rychlostní histogram a pomocí metody k-means nad parametry směru a rychlosti bodů stanovit požadovanou abnormalitu pohybu. Tato metoda nevyžaduje předzpracování obrazu pomocí separace objektů od pozadí, což ji činí ojedinělou.

Za abnormální pohyb je v tomto případě považována například odlišnost směru a rychlosti. Pokud se tedy skupina lidí pohybuje přibližně podobnou rychlostí, ale jedinec se pohybuje oproti skupině výrazně rychleji, je považován jeho pohyb za abnormální. Stejná situace je ještě zřejmější při využiti směru pohybu.

3.1 Hledání bodů zájmu

V každém snímku videosekvence je aplikován Harrisův rohový detektor postavený na autokorelační funkci, která měří lokální změny intenzity I v obraze s malou odchylkou w do různých směrů. Pomocí konvoluce je na obraz aplikována matice M o rozměru $2x2$ vypočítaná z derivátů intenzity obrazu v bodě (x, y).

$$M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \tag{10}$$

Funkce R následně vyhodnotí pomocí trasholdingu, zda se jedná o roh či nikoli a pozitivní výsledky zařadíme do seznamu výsledků.

$$R = detM - k(traceM)^2 \tag{11}$$

3.2 Sledování bodů zájmu

Body, detekované z jednotlivých snímků videa, je potřeba napříč snímky párovat. Jinými slovy potřebujeme vědět, kterému bodu ze snímku č. 2 odpovídá bod ze snímku č. 1. Tento problém je řešen pomocí Lucas-Kanade trackeru. Tento algoritmus promítne polohu bodu ze současného snímku do následujícího snímku a následně v blízkém okolím hledá odpovídající nový bod.

Tento přístup může v krajních situacích omezovat rychlost pohybujícího se objektu. Maximální pohyb bodu v rámci jednoho snímku pak totiž odpovídá velikosti prohledávaného okolí, což nemusí vždy dostačovat. Z tohoto důvodu je použit pyramidový L-K sledovací algoritmus, který tento problém řeší prohledáním obrazu postupným zmenšováním prohledávaného okolí. Díky tomu jsou zachyceny i body pohybující se větší rychlostí mezi jednotlivými snímky. Výstupem tohoto kroku jsou tedy vektory pohybu jednotlivých bodů mezi dvojicí snímků, přičemž směr vektoru udává směr pohybu bodu a délku vektoru můžeme chápat jako rychlost pohybu.

3.3 Sestavení prostoru vlastností

Na základě vztahů mezi dvojicemi snímků jsme schopni dále pracovat se souvislejší trajektorií jednotlivých bodů, protože ke spolehlivější detekci abnormality pohybu nám pouze dvojice snímků nestačí.

- frame-1:
  - point-1: $[x, y]$
  - point-2: $[x, y]$
  - point-n: $[x, y]$
- frame-2:
  - point-1: $[x, y]$
  - point-2: $[x, y]$
  - point-n: $[x, y]$
- frame-m:
  - point-1: $[x, y]$
  - point-2: $[x, y]$
  - point-n: $[x, y]$

Z tohoto důvodu je vhodné tyto vektory sloučit do prostoru, který pojme delší trajektorii bodů v čase. První dimenzi prostoru využijeme k označení snímku, nad kterým jsme provedli detekci. Druhá dimenze nám identifikuje nalezené body v dílčích snímcích a ve třetí uchováváme potřebnou vlastnost. Takovéto prostory můžeme sestavit pro polohy, vektory pohybů či zrychlení jednotlivých bodů.

Takovýto prostor nám pak dává možnost snadno analyzovat jednotlivé body v čase tím, že budeme pozorovat libovolný bod a jeho vlastnosti napříč první dimenzí. Tedy například porovnáním délky vektoru rychlosti určitého bodu v různých časech jsme schopni zjistit, zda byl pohyb bodu konstantní nebo zda zrychloval či zpomaloval a v jakém čase k tomu došlo.

3.4 Detekce abnormality pohybu

Ke zjištění abnormality pohybu použijeme metodu založenou na histogramu rychlosti. Pomocí prostoru vlastností sestavíme pro každý snímek histogram rychlostí jednotlivých bodů tak, že vodorovná osa bude odpovídat hodnotě rychlosti pohybu bodů a svislá osa bude znázorňovat, kolik bodů se pohybuje danou rychlostí. Takovýto histogram následně ještě upravíme tak, že posuneme krajní body osy rychlosti. Za nejmenší rychlost budeme tedy považovat místo nuly takovou rychlost, kterou se pohybuje bod v daném snímku nejpomaleji. Stejným způsobem pak určíme i opačnou hranici histogramu pro nejrychleji se pohybující bod.

Pokud budeme vycházet z tvrzení, že normální pohyb je takový, kterým se pohybuje skupina bodů, amplituda histogramu by v těchto rychlostech měla být větší než u bodů pohybujících se abnormálně.

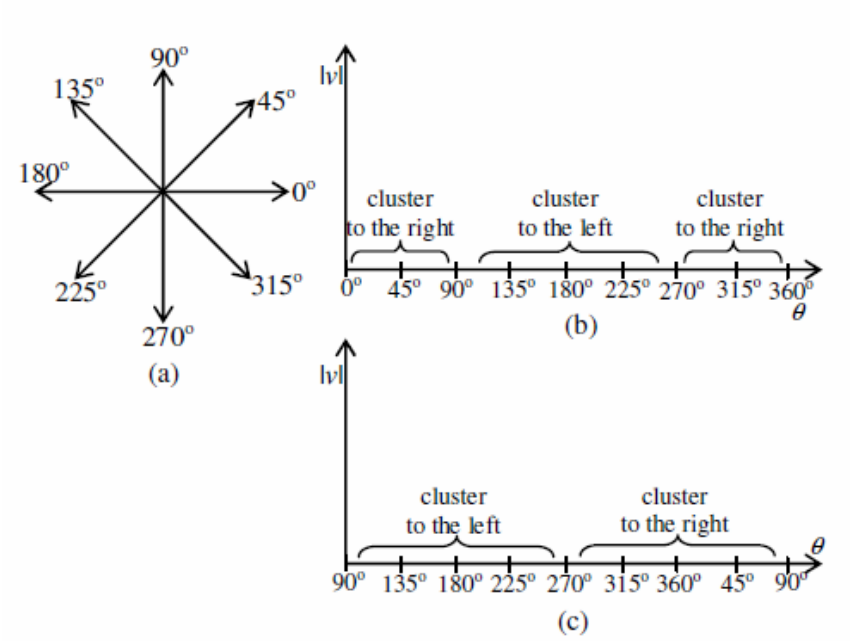
Můžeme také vyhodnocovat, zda je celá scéna abnormální. Tuto vlastnost zjistíme pomocí klasického trasholdingu. Ten budeme provádět nad rozdílem minimální a maximální hodnotou rychlosti pohybu v dané scéně. Následně stanovíme určitou referenční hranici, pomocí které rozhodneme, zda je celá scéna abnormální či nikoli, kde w je vypočtený koeficient normality.

$$scene = \begin{cases} normal & \text{if w} < threshold \\ abnormal & \text{if w} \geq threshold \end{cases} \tag{12}$$

3.5 Stanovení abnormálního směru

Další vlastnost, kterou lze dobře analyzovat, je směr pohybu bodů. K tomu nám pomůže metoda k-means, která pracuje na základě shlukování bodů s podobnými vlastnostmi. Tuto metodu aplikujeme na směr a zrychlení úsečky pohybu jednotlivých bodů. Ke správnému vyhodnocení je však nutné provést správnou korekci a posunout orientaci úsečky dle scény.

V článku, ze kterého vycházím, se pohybují lidé především ve vodorovném směru, a proto je důležité odlišit, zda se body pohybují doprava či doleva. Tomuto typu pohybu odpovídá následující korekce.

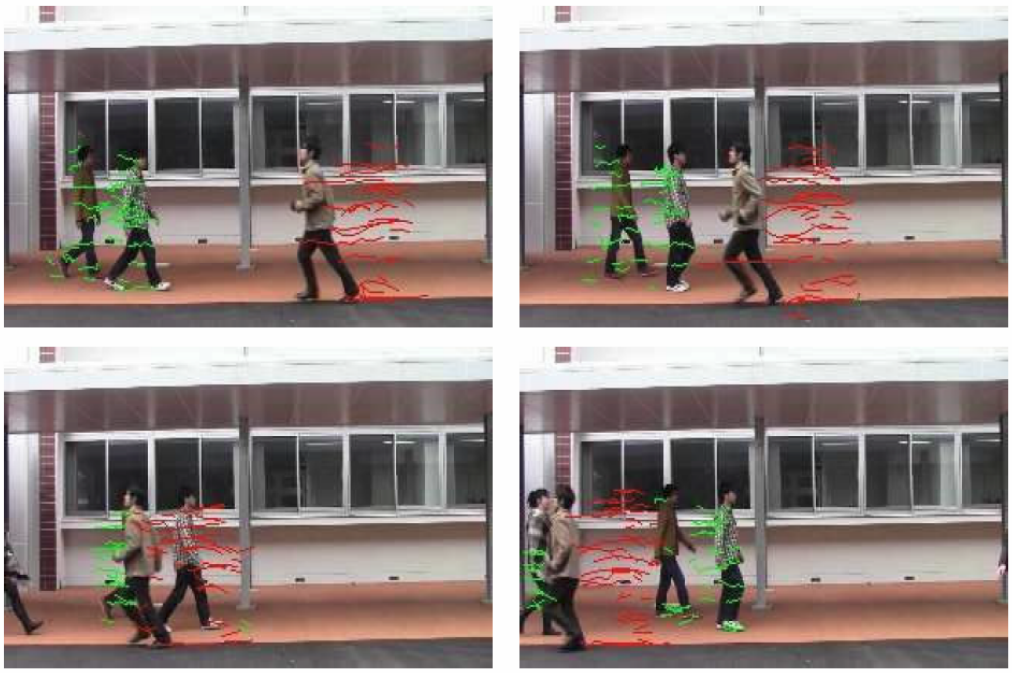


Obrázek 2: Korekce abnormality směru [1]

Pokud bychom však analyzovali záběr například ze silnice, kde se vozidla pohybují vůči obrazu svisle, není tato korekce potřeba. Naopak pokud je většina pohybu na obraze orientována pod nějakým jiným úhlem, je potřeba tomuto úhlu korekci přizpůsobit, aby metoda k-means správně fungovala.

3.6 Experimenty

Pro experimenty byly zvoleny záběry z vnitřních i venkovních prostor. Lidé se na nich pohybují normálně pomocí chůze či pomocí běhu, což reprezentuje abnormální pohyb. Rozlišení pořízených záběrů bylo 320x180 pixelů při 30 snímcích za sekundu. Analýza byla prováděna na hardwaru Intel® core™ i7 CPU 870 @2.93GHz, kde výpočet jednoho pozorovaného snímku trval 247ms a skládal se z 10 skutečných snímků pro zvýšení přesnosti. Na obrázku lze pozorovat červené čáry jako abnormální a zelené jako normální pohyb.



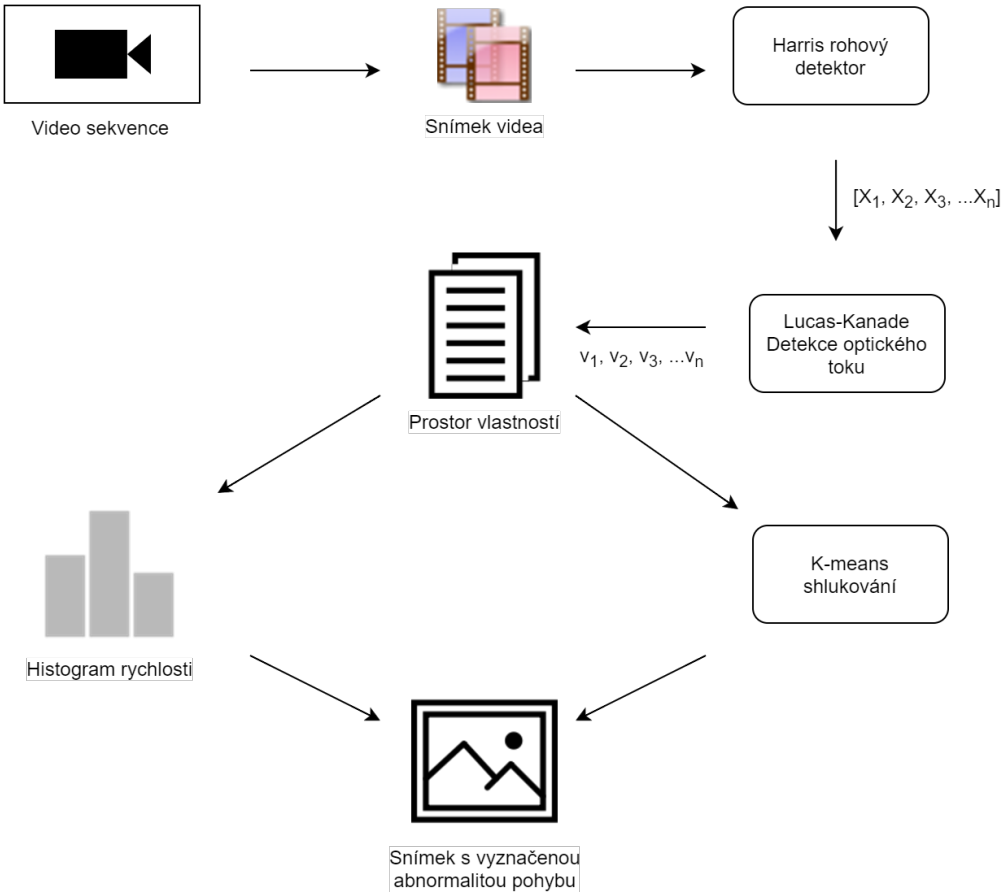
Obrázek 3: Výstup aplikace [1]

4 Implementace

Po nastudování problému a pochopení teoretické části problematiky jsem se rozhodl implementovat řešení popsané ve článku Abnormal Motion Detection in an Occlusive Environment [1] a vyzkoušet jej prakticky. Následně jsem se pokusil základní funkcionalitu realizovat jako škálovatelnou službu a poté metodu detekce vylepšovat. Implementaci jsem rozdělil do několika částí, které jsou popsány podrobněji v následujících kapitolách.

4.1 Návrh implementace původního řešení

Nejprve jsem se pokusil sestavit blokové schéma algoritmu s ohledem na co nejpřesnější shodu s postupem popsaným ve zmíněném článku.



Obrázek 4: Blokový diagram analyzátoru

Videosekvenci tedy zpracovávám po jednotlivých snímcích a předávám je Harrisově rohovému detektoru. Ten pomocí detekce dvou gradientů určí zajímavé body ve snímku a tyto informace předá Lukas-Kanade detektoru optického toku. Ten porovná souřadnice bodů aktuálního snímku s body z minulého snímku a vypočítá vektory pohybů těchto bodů. Následně se informace o kaž-

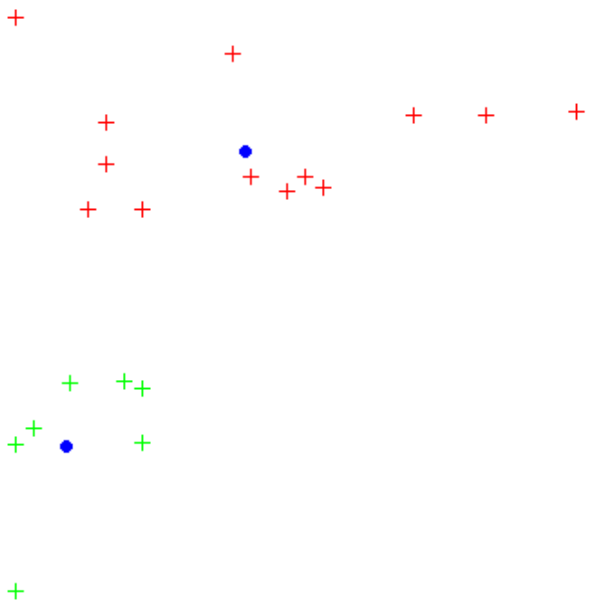
dém snímku uloží do prostoru vlastností, ze kterého se vypočítá pro aktuální snímek histogram rychlostí. Ten nám pomáhá určit abnormalitu rychlosti pohybu jednotlivých bodů.

Abychom byli schopni určit i abnormalitu směru, předáme prostor vlastností metodě K-means, která pomocí shlukování rozdělí body do dvou skupin podle abnormality jejich směru. Tyto skupiny bodů pak v obrázku vizualizujeme barvami a podle obou skupin. Výsledné obrázky pak spojím opět do videosekvence, kterou lze přehrát a pozorovat tak výsledek detekce.

4.2 Prototyp původního řešení

Pro usnadnění práce jsem využil pro realizaci prototypu knihovnu OpenCV. Ta mi umožnila využít Harrisův rohový detektor pomocí funkce *goodFeaturesToTrack* a Lucas-Kanade detektoru optického toku pomocí funkce *calcOpticalFlowPyrLK*. Následně jsem využil ještě funkce K-means pro výpočet shluků bodů a datové struktury matice pro snadnější manipulaci s obrázky.

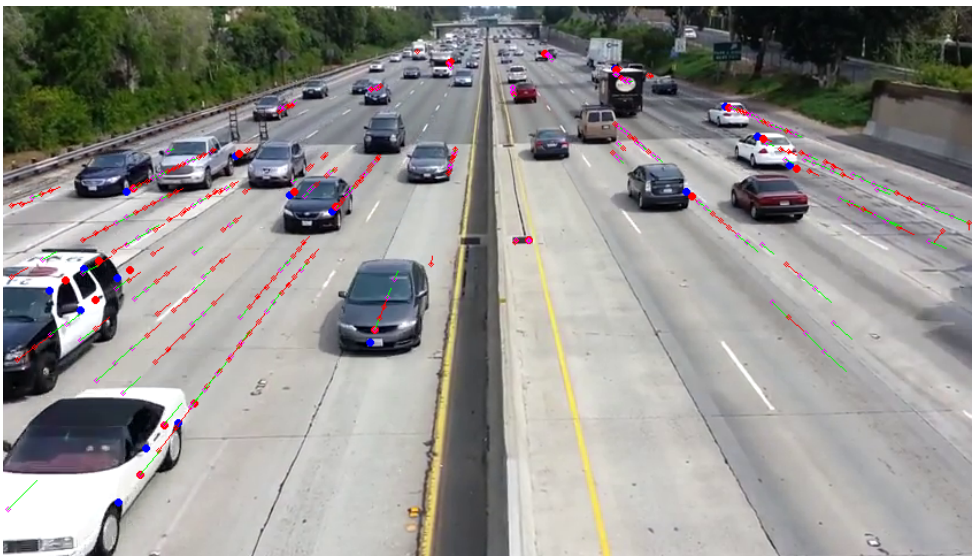
Pomocí těchto nástrojů jsem realizoval nejprve detekci zajímavých bodů obrazu a výpočet vektorů optického toku, které jsem uložil do prostoru vlastností pro každý snímek. Následně jsem vizualizoval rychlosti pohybu jednotlivých bodů do histogramu. Ten mi dal jako první informaci o abnormalitě pohybu, avšak vizualizovanou mimo samotnou videosekvenci. Dále jsem využil data z vektorového prostoru pro výpočet dvou shluků. Ty jsem označil barvami (normální pohyb zeleně a abnormální pohyb červeně) a vypočítal středy pro oba shluky, které jsou označené modrým kolečkem.



Obrázek 5: K-means klastrování

Pro lepší přehlednost jsem tyto informace spolu s vektory optického toku zobrazil do aktuálního analyzovaného snímku. Jednotlivá kolečka označují body, detekované Harrisovým rohovým

detektorem. Lucas-Kanade detektor optického toku našel mezi body dvou po sobě jdoucích snímků relace, které jsou zakresleny v obraze pomocí čáry. Tyto čáry jsou vykresleny vždy pro 20 předchozích snímků od aktuálního pro lepší vizualizaci samotného optického toku. Kolečka a úsečky jsou nakonec ještě obarveny stejnou barvou, jako shluky v obrázku výše, což vizualizuje abnormalitu pohybu. Vzhledem k tomu, že se auta na snímku pohybují v obou směrech podobnou rychlostí, je na trajektoriích vidět častá změna barvy o abnormalitě. Ta byla ovlivněna především aktuálním počtem aut na snímku v daném směru. Směr pohybu, který u aut převládal, byl proto označen za normální.



Obrázek 6: Výstup prototypu aplikace

Tento prototyp mi výborně posloužil k ještě hlubšímu pochopení problému a také mi reálně vizualizoval výsledky výpočtů přímo zpět do videosekvence. Experimentoval jsem také se vstupními konfiguračními konstantami jednotlivých detektorů a zjistil jsem, že je velmi obtížné nalézt nejvhodnější konfiguraci pro různé scény. Je proto třeba pro každou scénu provést alespoň experimentální kalibraci pro získání kvalitních výsledků. Narazil jsem zároveň na několik problémů, které se pokusím dále rozebrat.

Při záběrech z kamer, které snímají prostor pod určitým horizontálním úhlem, se projevují problémy apertury popsané v kapitole o optickém toku 2.2. Prakticky to znamená, že například auta, jedoucí po dálnici směrem ke kameře, se v obraze pohybují pouze malou rychlostí, ale zvětšují se. To způsobuje pro novější snímky výskyt nových zajímavých bodů, které se snažíme dále sledovat. Ve snímku, ve kterém tyto body detekujeme poprvé, však o nich nemáme v prostoru vlastností žádné předchozí záznamy, i když samotný objekt je ve videosekvenci rozpoznatelný již mnoho předchozích snímků. Objevují se tedy v obraze stále nové body zájmu, které dokážeme správně ohodnotit až v příštím a dalších snímcích. Dalším problémem jsou naopak mizející body zájmu, které náleží objektům, jež se svým pohybem posunuly za okraj zorného pole snímače. Ve vektorovém prostoru tak máme historii bodů k objektu, který už nelze v obraze pozorovat.

S těmito problémy se mi v rámci prototypu podařilo vypořádat, i když ne vždy tím nejvhodnějším způsobem. Nově detekované body například vizualizuji rovnou i přesto, že se jedná o falešně pozitivní detekci a hned v následujícím snímku tento bod již detekován není. Také trajektorie pohybů objektu zmizelých ze záběru kamery vizualizuji ještě nejvýše 19 následujících snímků, což může působit matoucím dojmem na pozorovatele. Tyto krajní situace by dle mého názoru měly být konfrontovány s konkrétní aplikací analyzátoru v reálném prostředí. Nelze tedy navrhnout jedno potenciálně správné chování za univerzální.

4.3 Škálovatelná implementace

Po úspěšné realizaci prvního prototypu jsem přemýšlel nad skutečnou aplikací takovéhoto softwaru a konfrontoval jsem je s aktuálními trendy při vývoji a distribuci softwaru. První myšlenkou byla aplikace na mobilních zařízeních. Ty mají k dispozici videokameru s kvalitním rozlišením obrazu a pořizují videosekvence s vysokým počtem snímků za sekundu. Bohužel však trpí zásadním problémem v podobě nedostatku výkonu. Vybraná zařízení by takovýmto požadavkům zřejmě vyhověla, ale pro ostatní by bylo nutné zvolit nemalé kompromisy v podobě rozlišení či počtu snímků, což by se od určité hranice výrazně projevilo na kvalitě detektoru. Nebylo by také možné udržet konzistentní parametry detektoru napříč různorodými zařízeními.

Opačnou myšlenkou té předchozí bylo tedy použití detektoru jako síťové služby. Tedy samotnou analýzu obrazu provádět na jiném zařízení, než probíhá pořizování záznamu scény. Takováto služba by tedy vystavovala do sítě pouze aplikační rozhraní (API), pomocí kterého může libovolné jiné zařízení se službou komunikovat a odesílat jí snímky videosekvence a očekávat odpovědi s výsledky. Nevýhodou takového řešení je například nemožnost ovlivnění kvality pořizované videosekvence, protože takováto služba je již pouze konzument pořízených dat. Naopak velkou výhodou je dostatečný výkon a snadné iterativní vylepšování aplikace bez nutnosti složité distribuce nových verzí. V případě, že by ve stejný okamžik chtělo službu využít velké množství zařízení najednou, lze snadno škálovat výkon aplikace hned na několika úrovních zároveň a úspěšně obsloužit všechny požadavky s minimálním zpožděním.

Vzhledem k výhodám a nevýhodám obou přístupů jsem se rozhodl pro realizaci druhé varianty. Z hlavních rysů tohoto řešení (dostatek výkonu a síťová komunikace) vyplývá, že se služba hodí pro provoz na cloudové infrastruktuře. Můžeme tedy hovořit o SaaS modelu.

4.3.1 CVaaS - Computer Vision as a Service

V současné době je velkým trendem návrh softwaru s ohledem na škálovatelnost a následný provoz na cloudové infrastruktuře. Tyto kroky lze sledovat například u společnosti Google, která se tímto směrem ubírá i v oblasti analýzy obrazu. Nově spustila svou službu Cloud Vision, která nabízí prostřednictvím veřejně dostupného aplikačního rozhraní službu pro detekci mnoha parametrů v obrázku, který je zaslán na vstupu.

Bohužel zatím nepodporuje analýzu videosekvencí a pokročilou detekci pohybu, což je vzhledem k mé práci škoda. Velmi rád bych totiž porovnal úspěšnost svého řešení právě s touto službou. Je ovšem zřejmé, že dostupnost těchto funkcí je pouze otázkou času a lze pozorovat neustálý vývoj kupředu v této oblasti.

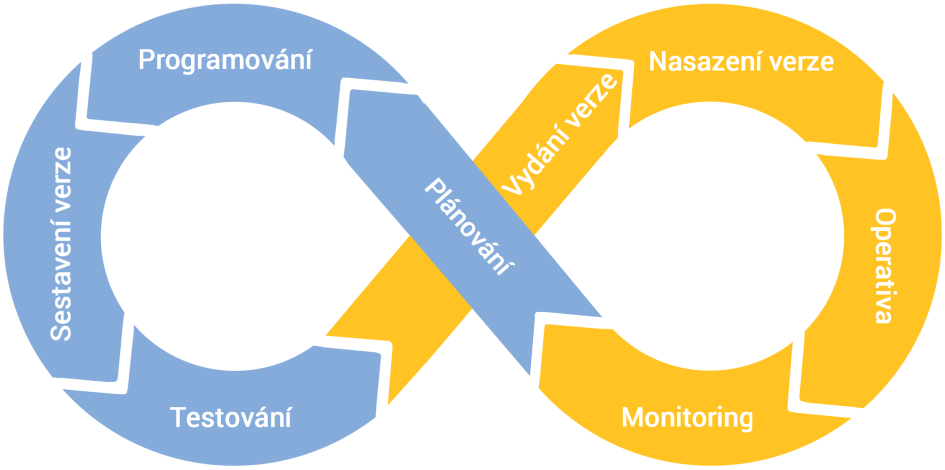
4.3.2 Technologický dluh

Technologickým dluhem můžeme nazvat stav systému (aplikace, infrastruktury, ...), kdy postupem času bylo přidáno mnoho nové funkcionality, ale neprováděla se údržba současného kódu (refactoring) či zanedbávala se tvorba dokumentace. Tento stav dlouhodobě vede ke špatné udržitelnosti a rozšiřitelnosti systému. V aplikaci se objevují duplicity v kódu či jednotlivé části aplikace mají příliš velkou zodpovědnost a funkcionalitu. Případné chyby se velmi špatně odhalují a odstranění takových chyb často vede k dalšímu zhoršení kvality kódu. Podrobnější informace o kvalitě kódu lze nalézt v publikaci The Clean Coder: A Code of Conduct for Professional Programmers [5].

Technickému dluhu lze dlouhodobě předcházet hned několika způsoby. Například automatizovanou statickou analýzou kódu, pokrytí klíčové funkcionality automatizovanými testy či aktualizací knihoven třetích stran a adaptací změn v jejich aplikačním rozhraní. Vybraným praktikám se podrobněji budu věnovat níže.

4.3.3 Vývojový cyklus

K zajištění udržitelnosti a rozšířitelnosti takovéto služby je potřeba zvolit správný proces vývoje. Tento proces by měl krom rozvoje aplikace iterativně odstraňovat technický dluh. Ten může službu postihnout hned v několika oblastech. Nejčastěji se jedná o špatnou organizaci, absenci automatizace sestavení projektu či chybějící nebo nedostatečné pokrytí klíčové funkcionality testy.



Obrázek 7: Vývojový cyklus aplikace

4.3.4 Plánování

Každý správný iterativní vývojový cyklus by neměl začínat programováním, ale správným plánováním a stanovením priorit pro danou iteraci. K tomuto účelu jsem využil nástroje Trello, kde jsem pomocí nástěnky (Board) rozdělil vývoj do mnoha úkolů. Tyto úkoly jsem dle jejich aktuálního stavu přesouval mezi pěticí sloupečků:

1. Ideas (Nápady)
2. Backlog (K realizaci)
3. Work in progress (Rozpracováno)
4. Quality Assurance (Testování)
5. Done (Hotovo)

Veškeré nové úkoly jsem vždy přidal do prvního sloupečku s nápady a zařadil vždy ty nejdůležitější nahoru. Následně jsem přesunul kartičky s největší prioritou do následujícího sloupečku "K realizaci", kde již byly úkoly, které skutečně budu realizovat. Opět jsem provedl seřazení dle priority a ten nejdůležitější úkol přesunul do sloupečku "Rozpracováno". V tomto sloupečku jsem se snažil udržovat vždy co nejméně kartiček, abych zajistil, že budu úkoly dotahovat do konce a nebudu mít mnoho rozpracovaných funkcí najednou.

4.3.5 Programování

Teprve po řádném plánování, které mi často nezabralo více jak pár minut, jsem se vždy pustil do nejdůležitějšího úkolu v daný moment a neztrácel čas realizací podružné funkcionality, kterou

sem sice chtěl realizovat, ale neměla aktuálně prioritu. Samotné programování pak nevyžadovalo příliš složitou analýzu, protože jsem celý problém rozdělil na malé dílčí části, čímž jsem si hrubou analýzu provedl. Následně již bylo potřeba zvolit jen správný přístup k implementaci. V případě, že jsem daný úkol dodělal, přesunul jsem úkol do sloupečku "Testování". V tuto chvíli jsem se mohl buď pustit do realizace dalšího úkolu nebo se pokusit otestovat právě implementovaný úkol. Takovýto proces by v běžném případě realizovalo více lidí najednou a testování by mohlo probíhat souběžně. Na této aplikaci však spolupracuji sám, a proto musím zastat i roli testera.

4.3.6 Sestavení verze

Díky automatizovanému procesu sestavení aplikace lze tento krok velmi snadno využít k urychlení vývojového procesu. Sestavení aplikace se často sestává z mnoha dílčích kroků. Ty, když se provádějí ručně, mohou vyústit v chybné sestavení například kvůli provedení kroků ve špatném pořadí či nechtěnému vynechání některého z kroků.

Automatizace v tomto kroku lze posunout ještě dál s využitím buildovacích nástrojů, jako je například Jenkins. Ty dokáží poslouchat výzvy verzovacího systému, stáhnout si vždy aktuální stav projektu a sestavit jej do podoby balíčku připraveného k distribuci. Například se sytémem GIT je vhodné spustit proces sestavení vždy, když proběhne PUSH nových COMMITů do hlavní větve projektu.

4.3.7 Testování

V případě, že jsem nenarazil na žádné chyby, mohu kartičku přesunout do sloupečku hotovo, v opačném případě jej vrátím do sloupečku Rozpracováno s doplněním komentáře o nalezených chybách. Ideální v této fázi vývoje je spuštění automatizovaných testů, které prověří kritické části aplikace. Vzhledem k tomu, že u mé aplikace nelze přesně stanovit kritickou cestu ke splnění cíle, psaní automatizovaných testů by bylo komplikované a příliš obecné. Přesto by ale dílčí části aplikace šly otestovat hned na několika úrovních.

4.3.8 Vydání a nasazení verze

Pomocí verzovacího systému GIT lze jednotlivé úkoly správně zařadit do hlavní stabilní větve, ze které lze vždy sestavit konkrétní verzi aplikace například do balíčku. Ten se při sestavení označí číslem verze, která se shoduje s TAGem ve verzovacím systému či je doplněna o počet commitů od poslední celočíselné verze. Podobným způsobem probíhá vývoj mnohých jiných projektů, ať už v komerční nebo open-source rovině. Po úspěšném sestavení se balíček nahraje do repozitáře, odkud jej lze snadno distribuovat na jeden či více serverů.

Tento způsob distribuce má velmi významnou výhodu v situaci, kdy je zjištěna v aktuálně běžící verzi chyba. Není nutné v takovém případě, aby systém běžel s chybou tak dlouho, dokud vývojář chybu neopraví. Lze snadno nasadit z repozitáře předchozí verzi, kde se chyba ještě nevyskytla a provozovat dále službu bez problémů. Souběžně mohou vývojáři chybu opravit

a připravit novou verzi, která již chybou postižena není. Základní funkčnost aplikace není ale po dobu opravy chyby ohrožena.

4.3.9 Operativa a monitoring

V této fázi je vhodné sledovat například množství požadavků na službu a zatížení systému. Tyto parametry lze porovnávat v jednotlivých denních, týdenních či měsíčních obdobích a navrhovat na základě zkušeností s provozem následné optimalizace či navýšení zdrojů infrastruktury. V případě kritické chyby může dojít i k výše popsanému nasazení starší verze a podobně. Úzce však tato fáze opět souvisí s plánováním, protože z reálného provozu lze nejlépe naměřit vhodná data, jako podklady ke zpřesnění plánování a určování priorit.

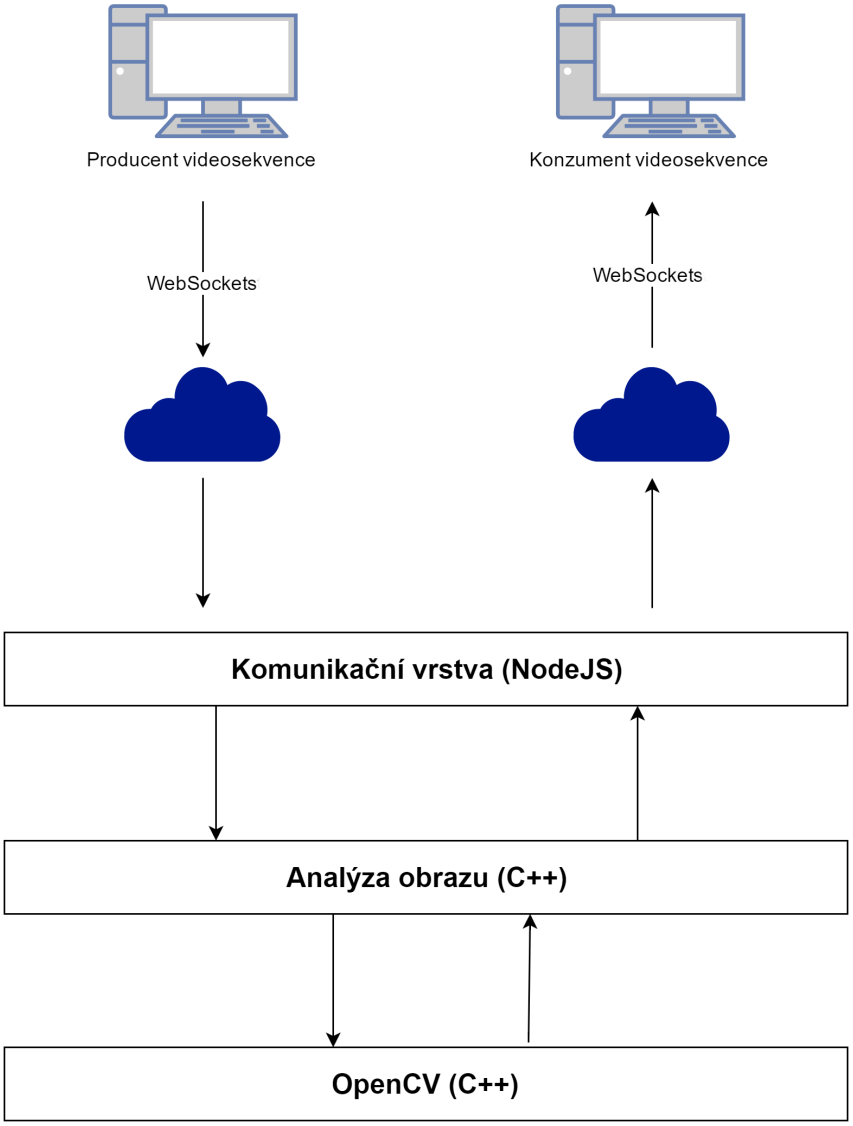
4.4 Architektura aplikace

Jak lze vidět na obrázku 8, aplikace je složena z několika důležitých částí, které mají zásadní vliv na její funkčnost. Pojďme si popsat jednotlivé celky podrobněji tak, jak putují data od zdroje až po výsledné zobrazení výsledků uživateli.

Vše začíná tenkým klientem, který odesílá obrazová data na server po vnitřní síti či internetu. Využívá k tomu technologii WebSockets, která je dnes podporovaná téměř každým moderním webovým prohlížečem. V našem případě je klient realizován jako malá HTML5 aplikace, která se stará o odesílání jednotlivých snímků na webový server. Může se jednat ale například i o chytrý telefon, IP kameru či například zařízení IoT (RaspberyPi, Arduino a další). Serverová vrstva také není nijak závislá na komunikaci přes WebSockets. Naopak je možné využít kombinaci jiných protokolů (WebRTC nebo klasickou socketovou komunikaci), které mohou být mnohem efektivnější a rychlejší.

Webový server, který je realizován pomocí technologie NodeJS, každý snímek dekóduje a předá jej další vrstvě. Díky tomu, že pracuje asynchronně, dokáže obsloužit velké množství požadavků, což přesně odpovídá očekávaným parametrům. Také je určen primárně k tvorbě webových aplikací, což velmi usnadňuje práci se síťovou komunikací. Oproti C++ nedosahuje sice takového výkonu, ale tuto nevýhodu plně převažuje svou jednoduchostí pro tento účel.

Další část aplikace můžeme považovat za výkonné jádro, ve kterém se ukrývá veškerá logika celého systému. Jedná se o nativní C++ modul do JavaScriptového enginu V8. To umožňuje interoperabilitu mezi oběma jazyky pomocí volání vlastních nativních funkcí z prostředí NodeJS. Kvůli výpočetní náročnosti analýzy obrazu je realizace pomocí nativního kódu obrovskou úsporou výkonu. Dále je také možné využít externí nativní knihovny, jako například použitá OpenCV. Tato knihovna se stará o dílčí výpočty nad obrazovými daty, které zvládá velmi rychle. Vrstva analýzy obrazu využívá dílčí části této knihovny pro zpracování každého snímku hned několikrát.



Obrázek 8: Architektura aplikace

Výsledky výpočtů z nejnižší vrstvy jsou vráceny analýze obrazu, které obohatí původní snímek o vypočtené informace a připraví další doplňující data. Řízení programu je pak spolu s výstupními daty předáno zpět do vrstvy se síťovou komunikací, které se postará o doručení dat konzumentovi. Takovýchto konzumentů může být hned několik na různých zařízeních nebo dokonce rozdílných částech světa.

V roli konzumenta je opět jednoduchá HTML5 aplikace, které také komunikuje se serverem pomocí WebSockets. Tuto část lze v případě potřeby také zcela nahradit. Aplikace má za úkol zobrazit uživateli přijatý snímek spolu s doplňujícími informacemi z výpočtů na serveru.

V případě, že budeme provozovat tyto službu na vnitřní síti, můžeme očekávat streamování obrazu v reálném čase s velmi dobrými parametry (v závislosti na výkonu serveru). Komunikace přes internet je limitována pouze objemem dat, který jsme schopni v reálném čase přenést, což

závisí na propustnosti linky mezi serverem a klienty. V případě, kdy běží producent i konzument na stejném zařízení, jsou požadavky na propustnost prakticky dvojnásobné. Můžeme v takové situaci očekávat mírné zpoždění při doručování jednotlivých snímků v obou směrech, což povede ke zpoždění obrazu na konzumentovi.

4.5 Infrastruktura

Vzhledem k tomu, že architektura aplikace je poněkud komplexní, je potřeba odpovídajícím způsobem přistoupit také k infrastruktuře, na které výsledná aplikace poběží. Tato infrastruktura je potřebná nejen pro běh výsledné aplikace, ale také pro její vývoj. Velmi často jsem se setkával u komplexnějších systémů s problémem, kdy ve vývojovém prostředí aplikace fungovala správně, ale na testovacím či produkčním prostředím se chovala odlišně a vykazovala chyby. Tento problém je primárně způsoben rozdílností těchto prostředí například v oblasti platformy, verze běhového prostředí nebo výkonu. Údržba tolika rozdílných prostředí je komplikovaná, a proto například i neaktuálností některé knihovny či aplikace může způsobit neočekávané chování celé služby.

Tuto situaci jsem se rozhodl řešit sjednocením všech prostředí do jediné konfigurace, kterou lze pouze s drobnými odlišnostmi automatizovaně nasadit na všechna prostředí. K tomu mi posloužilo hned několik nástrojů. Nejprve si pojďme ale říci něco o jednotlivých prostředích.

Vše začíná u vývojáře, který většinou vyvíjí aplikaci na svém rychlém notebooku s SSD diskem a velkým množstvím paměti RAM. Tyto úžasné parametry mu však velmi zkreslují představu o reálném prostředí, kde aplikace opravdu poběží. To je většinou u cloudového providera, který na jednom fyzickém serveru provozuje desítky či stovky virtuálních strojů. Ty se musí o fyzické zdroje navzájem dělit a na každý, i přes mnohonásobně výkonnější hardware, zbude jen zlomek výkonu, který má k dispozici vývojář na svém notebooku. V našem případě je cloudový provider Amazon Web Services, kterého se například týká omezení výkonu disku IOPS. Téměř stejné s produkčním prostředí by mělo být prostředí testovací. V tomto případě budu považovat testovací a produkční prostředí za identické, protože svou aplikaci nenasazuji do žádného ostrého provozu. Také z finančních důvodů by byl provoz dvou stejných serverů zbytečný pro účely této práce.

4.5.1 Automatizace infrastruktury

Základním stavebním kamenem správného přístupu k infrastruktuře je výše popsaná jednotnost prostředí. Klíčovými požadavky jsou proto hardwarové požadavky. Prakticky je ovšem nemyslitelné snažit se omezovat výkon pracovní stanice, pořízení staršího hardwaru a podobně. Dále je také potřeba provozovat aplikaci na stejném operačním systému, což je ještě větší překážka. Na serveru totiž často neběží ani grafické uživatelské rozhraní, bez kterého by nebylo možné zase spustit ani moderní webový prohlížeč.

Řešením tohoto problému je samozřejmě virtualizace. Ta umožňuje specifikovat nejen hardwarové požadavky, ale také provozovat odlišný operační systém na hostiteli i uvnitř virtuálního stroje. Konkrétně jsem použil aplikaci VirtualBox ve verzi 5 spolu s automatizačním nástrojem Vagrant, který mi umožnil snadno stroj konfigurovat a také provádět provisioning.

Instalaci virtuálního serveru jsem deklarativně popsal do YAML souborů pro skriptovací nástroj Ansible. Ten mi umožnil hned několik velmi důležitých věcí. Pokud bych například použil Bash a do scriptu popsal příkazy, které se mají při instalaci spustit, mohla by nastat situace, kdy budu chtít konfiguraci lehce změnit, nebo dojde k nějakému problému (například při aktualizaci). Opětovným spuštěním takového instalačního scriptu by mohlo dojít k neočekávaným situacím či konfliktům a spousta operací by se prováděla zbytečně (například restart služeb, jejichž konfigurace se nezměnila). Ansible díky deklarativnosti přistupuje k celému problému jinak. Popisuje cílový stav, ve kterém má celý systém být a provádí úpravy konfigurace jen tehdy, pokud se neshoduje s deklarovaným stavem. To umožňuje nedestruktivní opakované spuštění scriptů a pomocí takzvaných handlerů lze nadefinovat závislosti. Například pokud upravíme konfiguraci webového serveru, je potřeba jej restartovat. Restart služby se ale provede vždy jen v případě, že byla konfigurace změněna. Pokud se deklarovaný stav shoduje s reálným stavem služby, neprovede se nic. Další výhodou je možnost spustit konfiguraci na několika strojích zároveň, což umožňuje cloudovým službám velmi snadno škálovat. V mém případě mohu Ansible spustit na svém vývojovém prostředí (DevCloud), ale také na produkčním/testovacím serveru v AWS.

Díky tomuto přístupu mám jistotu, že vyvíjím na téměř shodném prostředí, jako je to produkční a minimalizuji tak pravděpodobnost výskytu chyb způsobených rozdíly v konfiguraci jednotlivých strojů. Pokud přeci jen dojde k nějakému problému v této oblasti, bývá většinou minoritní a jeho řešení obnáší pouze doplnění popisu stavu, do kterého má být stroj uveden. Vše se dá pak snadno nejprve otestovat na vývojovém prostředí, testovacím a nakonec na produkčním. Tento postup se nejlépe hodí aplikovat například na povyšování majoritních verzí klíčových služeb. Spousta problémů se tak odhalí již u vývojářů či testerů a minimalizuje se tak zavlečení chyby na produkční prostředí.

Ansible tedy pomáhá snadno automatizovat deployment infrastruktury a její průběžné udržování a správu. K jednotlivým položkám instalace (tasks) lze přiřadit štítky (tags), pomocí kterých lze napříč celou konfigurací seskupit určité úlohy do logických celků. Krom jiných (nginx, web, common, nodejs, ...) jsem přiřadil štítek "production"těm úlohám, které je nutné nainstalovat i na reálný server. Shoda produkčního a vývojového prostředí tedy není 100%, protože pro vývoj jsem potřeboval nainstalovat více nástrojů, které na produkci pak nejsou potřeba.

4.5.2 Hardwarová konfigurace hostitelského notebooku

Jedná se o laptop DELL XPS 15 s procesorem 6. generace Intel vyrobený 14nm technologií. Tato konfigurace obsahuje 4 jádra běžící na frekvenci 2.60 Ghz a možností krátkodobého přetaktování

Turbo Boost na frekvenci 3.50 Ghz. Operační systém byl použit Microsoft Windows 10 Pro 64-bit.

- Procesor Intel Core i7-6700HQ @ 2.60 GHz
- RAM: 16GB DDR4
- SSD: 512GB

4.5.3 Parametry virtuálního serveru

Pro virtualizaci jsem využil Oracle VM VirtualBox. Pro sdílení zdrojových kódů aplikace s hostitelským strojem jsem použil technologii NFS. Ta oproti výchozímu sdílení nabízí větší rychlost čtení i zápisu sdíleného adresáře, což mi umožnilo využít větší potenciál SSD disku hostitele.

- 2 procesorová jádra => odpovídá 1 fyzickému jádru reálného procesoru
- RAM 2GB
- SSD: 40GB

4.5.4 Parametry produkčního serveru

Jako poskytovatele produkční infrastruktury jsem zvolil jednoho z největších hráčů v této oblasti. Jedná se o Amazon Web Services. Z jeho obrovské nabídky služeb využívám pouze službu EC2, která nabízí využití VPS. Zvolil jsem jednu z nejnižších konfigurací t2.micro. Její parametry bohatě postačují testovacím účelům. V případě produkčního nasazení by však bylo nutné nasadit aplikaci na mnohem výkonější sestavu. Uvažovat bychom mohli také o několika méně výkonnějších serverech a předřazeném load balanceru.

- Typ: t2.micro
- CPU: 1x Intel Xeon Processors with Turbo up to 3.3GHz - 6 CPU credit/hour
- RAM: 1GB
- SSD: 8GB - 24 IOPS

4.6 Automatizace sestavení aplikace

Dalším velmi důležitým bodem je sestavení následný deployment aplikace. U malých aplikací se často setkáme pouze s jednoduchým procesem sestaveni a deployment není někdy třeba řešit vůbec. V případě této aplikace, které se skládá z mnoha dílčích částí a podslužeb je nezbytné mít sestavení aplikace plně pod kontrolou. Vynechání byť jediného kroku by mohlo znamenat

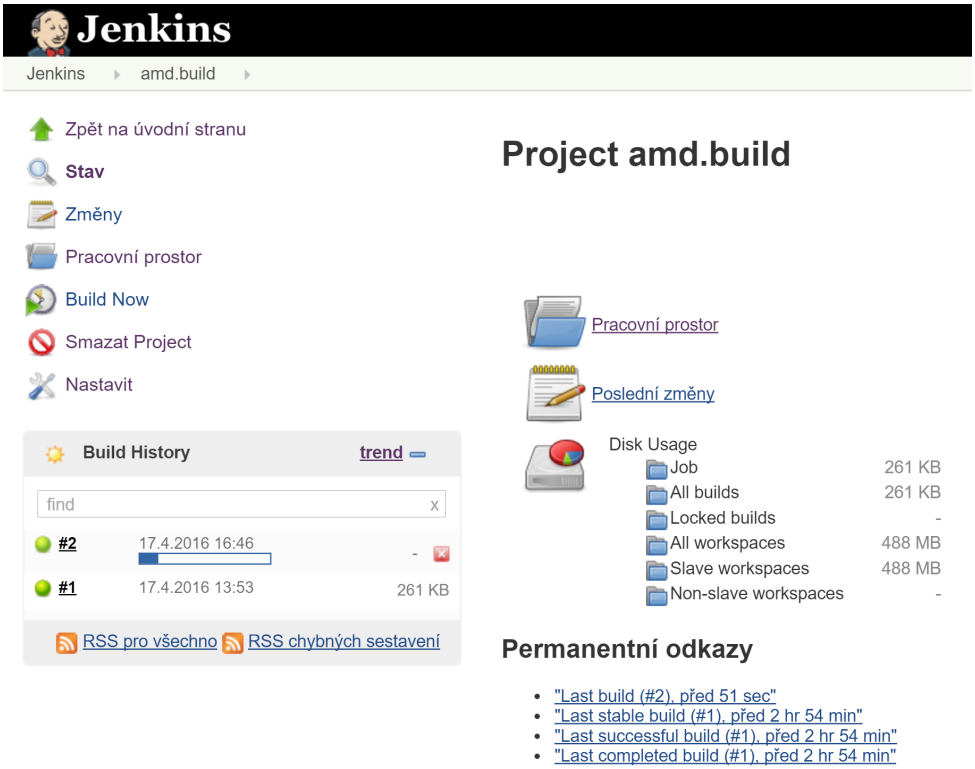
nefunkčnost celé služby, což si nemůžeme dovolit. Využil jsem proto open-source nástroje Jenkins, který tento proces výrazně usnadňuje.

Celou aplikaci se všemi komponentami zabalím do instalačního balíčku deb. Ten je řádně označen verzí a externími závislostmi. Jak bylo zmíněno výše, balíčkovací systém umožňuje snadno nasadit novější či starší verzi a zajistit tak minimální výpadek služby. Také lze nové verze aplikace velmi snadno distribuovat na rozsáhlou infrastrukturu. Poroces sestavení balíčku je podrobně popsán v souboru Makefile, který je v projektu ve složce *production*. Zde se také nachází preinstall a postinstall scripty, které zajišťují správné nasazení balíčku.

Číslo verze balíčku je vždy odvozeno jednotným algoritmem, který závisí na několika parametrech. Tím nejdůležitějším je *tag*. Tím chápeme číslo verze, které většinou pomocí stejnojmenné funkce ve verzovacím systému GIT označíme konkrétní commit. Takovýto štítek označuje stabilní verzi připravenou k vydání. Nejčastěji se používá sémantické verzování, které se skládá z trojice číslis oddělenými tečkou (1.0.5). První číslice se označuje jako $MAJOR$ a v případě změny verze na této pozici chápeme změnu jako zásadní. Může se jednat i o kompletní přepis některé části aplikace nebo například redesign. Přibývají nebo ubývají v těchto verzích zásadní části aplikace, které mají vliv na celkovou funkčnost. Druhá číslice se označuje jako $MINOR$ a je nositelem menších vylepšení a podružných změn funkcí aplikace. Poslední číslice je nazývána jako $PATCH$. V těchto verzích by se v aplikaci neměla měnit žádná funkčnost. Takto označená verze by měla primárně sloužit pro opravy současné funkcionality.

V případě, že v aplikaci nedošlo k žádným změnám od posledního tagu, název balíčku nese jako verzi pouze tento tag, což považujeme za stabilní sestavení. V případě, že následuje po posledním štítku alespoň jeden nebo více commitů, je přidána do verze balíčku ještě dodatečná informace o počtu commitů od poslední stabilní verze a kontrolní součet (hash). Takováto verze pak nese označení například $amd - app - v0.1.0 - 38 - gaaa0c90.deb$ a takovýto balíček je považován za vývojový a nestabilní. Nasazení takovéhoto balíčku pak může být provedeno pouze na vývojové a testovací prostředí. Často se tyto balíčky archivují do oddělených repozitářů, aby nedošlo například k nasazení vývojové verze na produkční prostředí.

Od sestavení aplikace, vytvoření balíčku s aplikací včetně všech aplikačních závislostí až po nasazení na vývojové prostředí má pod kontrolou nástroj Jenkins. Ten má definován takzvaný *Job*, který se v našem případě jmenuje *amd.build*. Krom samotného sestavení Jenkins poskytuje také podrobné statistiky o délce trvání sestavení a pomocí mnoha pluginů dovede velmi spolehlivě spočítat i mnoho metrik nad zdrojovým kódem aplikace. Těmito metrikami mohou být například statická analýza kódu, kontrola konvencí či spouštění automatizovaných testů. Na vývojové infrastruktuře je tento nástoj k dispozici na adrese https://jenkins.devcloud.local.



Obrázek 9: Jenkins

4.7 Klientské aplikace

Jak již bylo zmíněno, aplikace se skládá z mnoha oddělených celků a podslužeb. Pro uživatele jsou však nejzajímavější dvě z nich, které tvoří uživatelské rozhraní pro produkci a konzumaci videosekvence. Tyto aplikace nazýváme klientské, protože jsou nezávisle na směru toku dat obě konzumenty vystaveného aplikačního rozhraní serverové části. Podrobně si popíšeme fungování obou znich a vysvětlíme základní prvky uživatelského rozhraní.

Tyto samostatné aplikace jsou realizovány pomocí rodiny technologií HTML5, tedy HTML, CSS a JavaScript. To umožňuje aplikacím běh v mnoha moderních prohlížečich nezávisle na platformě či hardwaru.

Pro přehlednější organizaci kódu jsem zvolil preprocesor SASS pro CSS a BabelJS pro ECMAScript 2015/ECMAScript 6. Tyto transpilery mi umožnily skládat aplikaci mnohem modulárněji a sdílet dílčí části kódu s jinými částmi celé služby.

V případě vizuálního rozhraní jsem použil velmi rozšířený framework Bootstrap od tvůrců sociální sítě Twitter ve verzi $4 - alpha2$. Díky tomu, že je také realizován pomocí jazyka SASS, jsem využil pouze konkrétní části, což mi pomohlo minimalizovat celkový objem kaskádových stylů potřebných ke správnému běhu aplikace.

Veškerá funkcionalita aplikací je realizována pomocí ECMAScript 2015. Díky novým konstruktům tohoto jazyka, jako například iterátory, práce s poli nebo moduly, je kód mnohem

přehlednější a snadněji udržovatelný. Vzhledem k tomu, že podpora webových prohlížečů ještě není plně kompatibilní s celou specifikací ES2015, musel jsem využít transpilátoru, který převedl tento kód do staršího standardu ECMAScript 5.

Využití transpilátorů mi sice umožnilo použít novější standardy jazyka a pokročilé funkce, přibyla mi však režije v podobně samotné transpilace. Tu je potřeba provést pokaždé, když dojde k jakékoli změně v *.scss* či *.es*6 souborech. Jedním z řešení je konfigurace vývojového prostředí, které dokáže tuto transpilaci provádět vždy po uložení daného souboru. Problém je však v tom, že se takovýto proces nedá snadno zopakovat při sestavení aplikace například pomocí nástroje Jenkins. Každý další vývojář, který by pak chtěl na aplikaci spolupracovat, by musel složitě konfigurovat své vývojové prostředí a mohlo by se také stát, že by dokonce preferoval zcela jiné nástroje, které by mu neumožňovaly stejnou konfiguraci. Tento postup tedy vede k nekonzistenci hned na několika úrovních, a proto jsem se rozhodl raději zvolit alternativu.

Vhodnější možností je v tomto případě popsání veškerých kroků do jediného konfiguračního souboru, který se vždy postará o správné sestavení výsledných *.css* a *.js* souborů. K tomu jsem zvolil široce používaný nástroj Gulp. V souboru *gulpfile.coffee* je popsaných několik úkonů složených dohromady pomocí závislostí tak, aby se vše provedlo ve správném pořadí a zároveň se mohly provádět dva úkony paralelně.

```
-- Running Gulp tasks
cd .. &&\
            node_modules/.bin/gulp production --appVersion=v0.1.0-39-gdeb6d1e
[16:47:06] Requiring external module coffee-script/register
[16:47:07] Using gulpfile ~/jobs/amd.build/workspace/app/gulpfile.coffee
[16:47:07] Starting 'production'...
[16:47:07] Starting 'copy'...
[16:47:07] Finished 'production' after 13 ms
[16:47:07] Finished 'copy' after 39 ms
[16:47:07] Starting 'css'...
[16:47:07] Starting 'sass'...
[16:47:07] Starting 'js'...
[16:47:07] Starting 'transpile'...
[16:47:08] Finished 'sass' after 981 ms
[16:47:08] Starting 'cssmin'...
[16:47:08] Version: webpack 1.13.0
          Asset     Size  Chunks             Chunk Names
  app-camera.js  14.3 kB       0  [emitted]  app-camera
app-receiver.js  10.1 kB       1  [emitted]  app-receiver
[16:47:09] Finished 'transpile' after 1.66 s
[16:47:09] Starting 'uglify'...
[16:47:09] Finished 'cssmin' after 686 ms
[16:47:09] Finished 'css' after 1.67 s
[16:47:09] Finished 'uglify' after 276 ms
[16:47:09] Finished 'js' after 1.94 s
```

Obrázek 10: Sestavení HTML5 aplikace pomocí nástroje Gulp

Jak lze vidět na obrázku 10, vše začíná spuštěním úlohy *production* s parametrem $appVersion$. Ta spustí další úlohy *copy*, *css* a *js*. První úloha je uvedena jako blokující, takže další dvě musí počkat, až se první dokončí. Zbytek úloh však ale může běžet souběžně, protože se nijak navzájem neovlivňují. Úloha *copy* má na starosti zkopírování externích závislostí do adresáře s aplikací,

aby nebylo nutné do výsledného balíku vkládat celý repozitář závislosti včetně příkladů použití a dokumentace.

Následuje spuštení úlohy *css*, která nejprve provede transpilaci souborů *.scss* na *.css*, a následně provede minifikaci. Díky tomu mají výsledné styly mnohem menší objem, protože jsou odstraněny všechny nepotřebné komentáře a bílé znaky. Při minifikaci je proveden ještě jeden důležitý krok. Pomocí nástroje Autoprefixer jsou doplněny vendor prefixy pro novější CSS vlastnosti. Konfigurace tohoto nástroje je provedena pomocí speciálních řetězců, které relativně i absolutně popisují prohlížeče, na které chceme cílit.

$$'last2version','safari5','ie9','opera12.1','ios6','android4'$$

Podobný postup je aplikován i na JavaScript pomocí úkolu *transpile*. Ten spustí nástroj WebPack, který zajistí poskládání jednotlivých ES2015 modulů do jediného. Dále se spustí úloha $uglify$, která pomocí stejnojmenného nástroje provede opět minifikaci JavaScriptu. Výsledné soubory jsou pak přibaleny do finálního balíčku a originály smazány.

Takovýto postup lze pomocí jediného příkazu spustit jak při procesu sestavení aplikace, tak i při vývoji. Tím zajistíme konzistenci a správnost sestavení nezávisle na prostředí, kde zrovna vývojář pracuje. Aby však nebylo při vývoji nutné Gulp spouštět opakovaně při úpravě zdrojových souborů, připravil jsem ještě jednu úlohu navíc. Jmenuje se *watch* a má za úkol při prvním spuštění monitorovat konkrétní adresáře a soubory a při jejich změně spustit celý výše popsaný proces automatizovaně.

Další výhodou těchto transpilátorů je statická analýza zdrojového kódu. Vzhledem k tomu, že JavaScript je interpretovaný jazyk, vývojář se například i o syntaktické chybě dozví až při spuštění aplikace. To je velmi nepohodlné a zdlouhavé. Díky preprocesingu je však informován o spoustě problémech okamžitě a výrazně to zefektivnějuje jeho práci.

Vzhledem k tomu, že obě aplikace nejsou hlavním cílem této práce, nevěnoval jsem jejich optimalizaci a grafické přívětivosti příliš mnoho času. Také jsem je testoval především v prohlížeči Google Chrome verze $49.0.2623.112m$, takže se mohou vyskytnout drobné problémy v ostatních verzích či prohlížečích. Kladl jsem naopak důraz na realizaci klíčové funkcionality, automatizace řešení a vzdálenou konfigurovatelnost serverových výpočtu. V případě nasazení aplikace do reálného provozu je však nezbytně nutné tyto kroky realizovat.

4.7.1 Producent videosekvence

Hlavní princip této aplikace spočívá ve zpracování jednotlivých snímků videosekvence, které následně odesílá spolu s doplňujícími konfiguračními daty na server pomocí technologie WebSockets. Zdrojů snímků však může být více. Primárním zdrojem jsou dílčí obrázky, které uživatel načte z lokálního počítače. Obrázky mohou být ve formátu JPG, PNG nebo GIF. Tyto soubory musí obsahovat v názvu souboru unikátní číslici, pomocí které dojde následně k seřazení. Tento

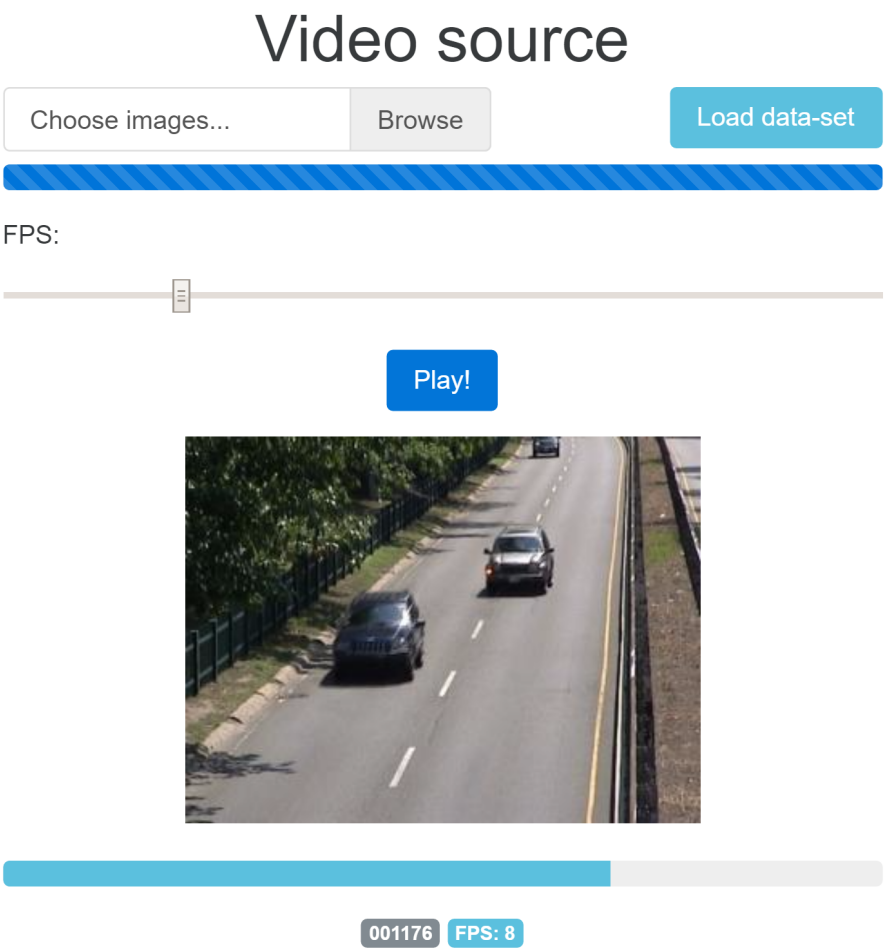
krok je velmi důležitý, aby proběhlo přehrávání videosekvence korektně a následná analýza obrazu nevykazovala nesmyslné anomálie.

Druhým zdrojem může být připravený testovací data-set uložený na serveru. Ten lze snadno načíst pomocí tlačítka "Load data-set". Videosekvence často obsahuje velké množství snímků (ukázkový data-set jich obsahuje 1700), z toho důvodu je načítání obrázků z obou zdrojů signalizováno modrým vodorovným teploměrem. Po úspěšném načtení si dále může uživatel zvolit rychlost, jakou se videosekvence bude přehrávat a odesílat data na server. Ve výchozím stavu je kvůli plynulosti zvoleno 20 snímků za sekundu. Limitní se v případě provozu cloudového provozu může stát propustnost internetové linky.

Snadným výpočtem s několika předpoklady si dovedeme spočítat realistický výkon datového přenosu. Předpokládejme tedy dle rozlišení výchozího data-setu $320x240$ bodů, že každý snímek bude mít v průměru $15KB$.

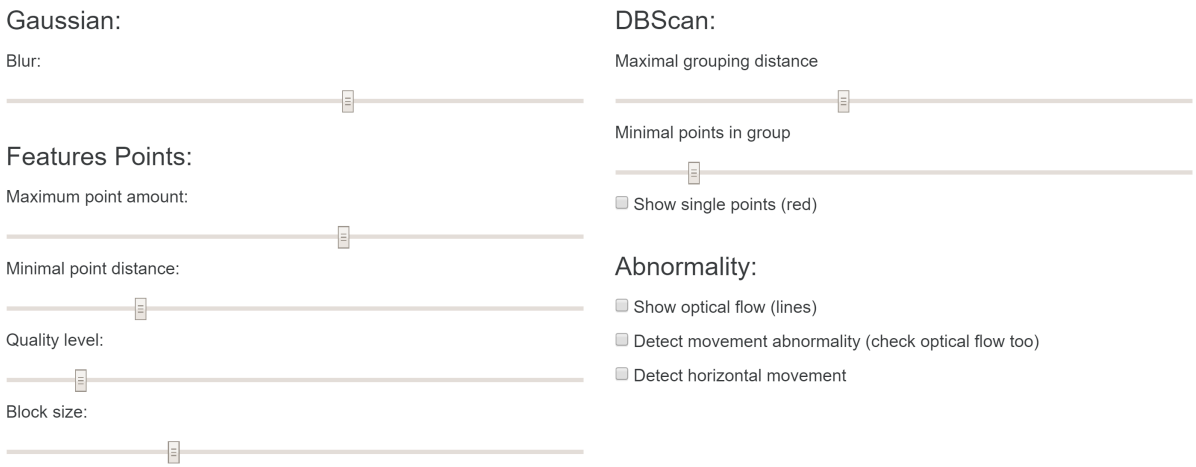
$$15KB * 20FPS = 300KBps \approx 2400Kbps = 2,4Mbps$$

Hodnota $2,4Mbps$ je kapacita, kterou v dnešní době zvládne většina linek bez větších obtíží. Přesto lze velmi snadno lehkým snížením rychlosti docílit plynulého přehrávání a odesílání videosekvence i na slabší konektivitě. Podrobnosti o vylepšení datového přenosu jsou rozebrány v kapitole Možnosti dalšího vylepšení 4.9.



Obrázek 11: Producent videosekvence

Výše popsané prvky uživatelského rozhraní slouží k základnímu ovládání aplikace. Pod náhledem přehrávané videosekvence se však nacházejí další volby pro konfiguraci serverové části služby. Tyto volby mají přímý dopad na kvalitu analyzátoru a před každou analýzou by mělo dojít alespoň k experimentální konfiguraci těch nejdůležitějších. Přesné hodnoty jednotlivých parametrů jsou při každé změně vypsány do vývojářské konzole. Pokusím se ve stručnosti tyto volby popsat v následujícím textu.



Obrázek 12: Vzdálená konfigurace analyzátoru

4.7.1.1 Gaussian V této sekci lze ovládat preprocesing. Ten se provádí aplikací pomocí aplikace konvoluční masky na každý snímek videosekvence. Jak samotný název napovídá, masku tvoří Gaussian. Pomocí táhla můžeme měnit úroveň rozmazání. Té je docíleno tak, že se mění velikost konvoluční masky v obou rozměrech zároveň. Výsledek lze pozorovat na černobílém snímku na konzumentovi. V případě, že posuneme táhlo zcela vlevo, dojde k deaktivaci preprocesingu. Tato volba se může hodit především pro videosekvence s málo výraznými konturami, kterým by rozmazání ještě více zhrošilo detekci hran či významných bodů.

4.7.1.2 Features Points Tato část ovládacích prvků je poněkud rozsáhlejší, jedná se o konfiguraci Lucas-Kanade detektoru významných bodů. Tento krok analýzy je velice důležité zkonfigurovat správně, protože k následující úspěšné detekci je potřeba získat dostatek kvalitních bodů z každého snímku.

První volbou je "Maximum point amount". Toto tato hodnota určuje nejvyšší možné množství detekovaných bodů. Detektor na výstup nezašle vždy odpovídající počet bodů, který uživatel nastaví. Nikdy však nastavené množství nepřekročí. Tato volba se hodí v situacích, kdy máme v obraze velké množství rohů a hran. Můžeme pak snadno tyto body shora omezit.

Pomocí volby "Minimal point distance"můžeme ovlivnit, jak blízko sebe mohou detekované body být nejblíže. Díky tomu můžeme ovlivnit rovnoměrnost rozprostření bodů na objektech. Pokud je tato vzdálenost nastavena na příliš nízkou hodnotu, mohou se všechny body vyskytnout na nejvýraznějším z objektů, přičemž další, méně výrazné objekty, již sledovány nejsou. Je tedy vhodné minimální vzádenost mezi body navýšit, čímž dosáhneme vždy pouze limitního počtu detekovaných bodů zájmu na objektech určité velikosti.

Prvek "Quality level"vyjadřuje míru přesnosti, s jakou jsou hodnoceny rohy v obraze. Považujme tuto hodnotu za práh, kterým jsou filtrovány nalezené body dle kritéria výraznosti. V případě, že máme málo výrazné hrany ve videosekvenci, je vhodné tuto hodnotu lehce snížit,

čímž získáme větší množstvé méně přesných bodů. Následné shlukování nám pomůže tyto body opět zpřesnit, čímž zajistíme lepší kvalitu výsledné analýzy.

Poslední volbou této části je "Block size". Tato volba určuje velikost oblasti Ω, neboli okolí, na které se pomocí konvoluce hledají významné body. Tato volba ovlivňuje zároveň oba rozměry okolí, protože použití oblasti ve tvaru čtverce je nejčastější. Více o fungování této metody se lze dočíst v kapitole Harris corner detector.

4.7.1.3 DBScan První parametr "Maximal grouping distance"říká algoritmu DBScan maximální možnou vzdálenost dvojice bodů uvnitř skupiny. Největší vliv na clustering má v tomto případě velikost samotného objektu, respektive vzdálenost jednotlivých detekovaných bodů. Parametr předchozí části "Minimal point distance"by měl být vždy menší než tento. V opačném případě bude detektor nalézat body vždy ve větší vzdálenosti, než je potřeba pro vytvoření skupiny a všechny nalezené body budou díky tomu označeny za šum.

Druhým parametrem je "Minimal points in group". Tento parametr určuje práh nejmenšího možného počtu prvků tvořících skupinu. Ve výchozím stavu je tento parametr nastaven na hodnotu 2, což pomáhá snadno eliminovat většinu sirotků, ale zároveň umožňuje detekci i malých objektů. Posunutím tohoto táhla zcela doleva dojde k deaktivaci shlukování. Tuto volbu je vhodné aplikovat například na videosekvence s mnoho malými objekty.

Poslední volba "Show single points"nijak neovlivňuje výpočty analyzátoru. Jedná se pouze o zapnutí nebo vypnutí vizualizace jednotlivých bodů před provedením shlukování. Tuto volbu je vhodné použít při experimentální konfiguraci pro snazší rozhodnutí, zda využít klastrování či nikoliv. Vstupní body shlukování jsou na výstupu obarveny červenou barvou.

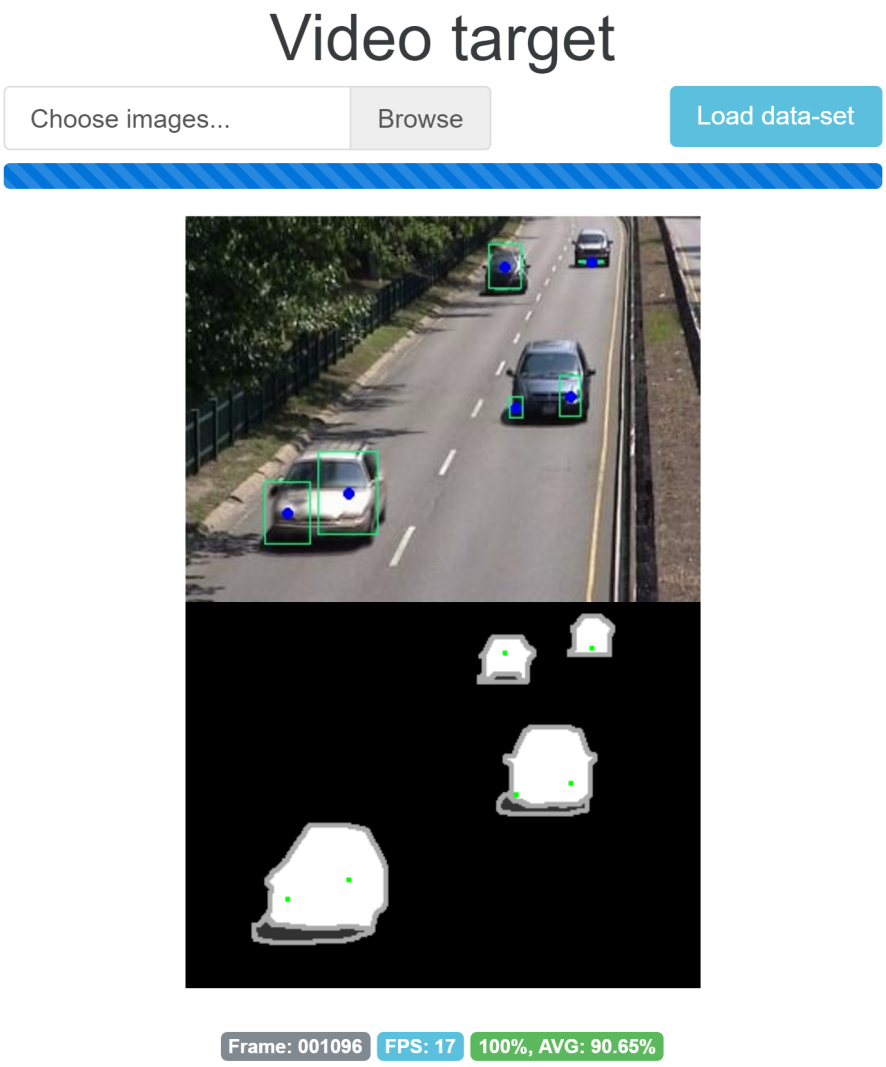
4.7.1.4 Abnormality Tato skupina voleb se skládá ze tří dvoustavových zatržítek. První z nich "Show optical flow"neovlivňuje výpočty, ale zapíná nebo vypíná zobrazení historie až 30 posledních pohybů sledovaného bodu ve scéně. Tato historie je na výstupu vizualizována zelenou čarou v podobě určitého ocasu za každým bodem. Lze tak snadno pozorovat plynulost pohybu objektu, trasu a pomocí délky jednotlivých úseček i rychlost pohybu. Tato vizualizace je výsledkem výpočtu optického toku ze středů skupin bodů nebo přímo z dílčích bodů zájmu.

Následující volba "Detect movement abnormality"umožňuje uživateli zapnout detekci abnormality pohybu. Každý pohyb bodu mezi dvěma snímky je pak rozdělen pomocí shlukování K-means do jedné ze dvou skupin. Ta větší vždy reprezentuje pohyb normální a menšina zase pohyb abnormální. Vizuálně je abnormalita zobrazena na výstupu obarvením úseček pohybu objektů červenou barvou.

Třetí volba "Detect horizontal movement"má na starosti korekci abnormality směru. Pokud se objekty ve scéně pohybují vůči snímači ve svislém směru, je vhodné tuto volbu nechat vypnutou. Pokud se však ve scéně pohybují objekty spíše po vodorovné ose, je nutné tuto změnu zahrnout do výpočtu.

4.7.2 Konzument výsledných dat

Tato aplikace je do značné míry pouze vizualizací výpočtů, které proběhnout na serveru. Jedinou vyjímkou je možnost provedení testu úspěšnosti služby. Podrobně si tedy projděme hlavní části uživatelského rozhraní a princip fungování testu úspěšnosti.



Obrázek 13: Konzument výsledných dat

Stejně jako u aplikace producenta videosekvence obsahuje v horní části uživatelské rozhraní volby pro načtení vlastního datatsetu či načtení výchozího. Tyto volby jsou určeny pro test úspěšnosti a nejsou potřeba pro pouhou vizualizaci výsledků. Následuje nejdůležitější část aplikace, kde jsou zobrazovány jednotlivé snímky videosekvence tak, jak jsou přijaty ze serverového aplikačního rozhraní opět pomocí technologie WebSockets. Tyto snímky jsou však obohaceny o vizuální reprezentaci výpočtů analyzátoru. Lze tedy snadno pozorovat detekované objekty a vizualizaci pohybu v závislosti na nastavení voleb producenta.

Pro lepší zpětnou vazbu při konfiguraci jednotlivých voleb jsem přidal ještě několik dalších prvků. Prvním z nich je náhled scény v odstínech šedi, kde lze pozorovat úrovně rozmazaní při preprocesingu a provést vizuální kontrolu obrazu, ze kterého bude následně provedena detekce bodů zájmu. Dalším prvkem je jednoduchý histogram, který se pojí s vyhodnocováním abnormality pohybu. Zobrazuje do sloupečků množství výskytu pohybu určitou rychlostí. Pokud se tedy pohybuje několik bodů stejnou rychlostí, utvoří v histogramu jeden vysoký sloupeček. Pokud se v obraze pohybuje mnoho bodů různými rychlostmi, jsou zobrazeny v histogramu jako mnoho malých sloupečků. Tyto sloupečky jsou seřany vždy podle rychlosti a jejich vzájemný odstup určuje rozdílnost rychlosti pohybu. Rychle pohybující se objekty najdeme v histogramu vpravo a pomalý pohyb zase vlevo.

V neposlední řadě při zapnuté volbě pro detekci abnormality pohybu jsou na posledním obrázku zobrazeny 2 shluky bodů společně s jejich středy. Použité barvy opět odpovídají výše popsaným stavům. Lze tak snadno pozorovat míru odlišnosti obou skupin na základě obou hodnotících parametrů a také míru relevance příslušnosti jednotlivých bodů do dané skupiny.

4.7.2.1 Testování úspěšnosti V případě, že uživatel načte jedním ze dvou uvedených způsobů již předem označená data pomocí jiného detektoru, lze snadno nechat tuto metodu otestovat. Vzhledem k tomu, že nelze snadno určit etalon v této disciplíně, je test úspěšnosty vždy pouze tak relevantní, jak kvalitní data jsou mu poskytnuty jinou metodou. Z tohoto úvodu jsem test doplnil i o vizuální kontrolu pro uživatele, aby mohl i subjektivně zhodnotit celou situaci.

Vstupní snímky by měly být pojmenovány stejným jménem souboru, nebo alespoň obsahovat v názvu unikátní číslici, pomocí které bude možné oba snímky spárovat. Díky tomu lze ověřit, zda bod, který tato služba označila za pohybující se objekt, byl označen stejným způsobem i jiným algoritmem. Porovnání pak probíhá na principu porovnání rozdílu barvy pixelu na konkrétních souřadnicích. Jinými slovy, pokud je na souřadnicích službou detekovaného bodu černá barva, jedná se o chybnou detekci, protože jiný detektor takto označil statické pozadí. V případě, kdy se jedná o nějakou úroveň šedi, jedná se pravděpodobně buď o obrys nebo přímo o pohybující se objekt. Detekce takového bodu je tedy správná.

Pro vizualizaci výsledků testování jsem proto zvolil zobrazení snímku jiného algoritmu ihned pod vizualizaci mého detektoru. Dále jsem do tohoto snímku zakreslil úspěšně detekované body zelenou barvou a falešné detekce barvou červenou. Následně proběhne výpočet okamžité úspešnosti pro každý snímek, jako počet úspešně detekovaných bodů $hits$, k celkovému počtu detekovaných bodů all.

$$hits/all = success \tag{13}$$

Stejným způsobem je pak spočítána i průměrná úspešnost napříč celou videosekvencí. Obě tyto informace jsou procentuálně vyjádřeny v zeleném políčku pod oběma snímky.

4.8 Serverová část služby

Tato část služby tvoří opět samostatnou aplikaci, která komunikuje s ostatními částmi pomocí definovaného aplikačního rozhraní. Mohli bychom tuto aplikaci rozdělit ještě do dvou nezávislých celků, ale kvůli zjednodučení za účelem této práce tvoří tyto části jeden celek.

4.8.1 Distribuce klientských aplikací

První částí je podpora pro zmíněné klientské aplikace. Jedná se v podstatě o webový server, který má na starost distribuci statických souborů pro oba klienty. Jedná se nejen o HTML soubory, ale také o CSS a JS soubory a v neposlední řadě také mnoho obrázků z předpřipraveného data-setu. O tuto práci se stará převážně webový server Nginx. Nejprve jsem na vývojovém prostředí používal distribuční větev *stable*, která poskytovala tento software ve verzi 1.8. Tato verze obsahovala rozšíření SPDY od společnosti Google, které dokázalo na rozdíl od klasického protokolu HTTP při velkém množství požadavků navázat pouze jediné spojení mezi klientem a serverem a pomocí něj všechny zpracovat. Dále s pomocí komprese došlo i ke snížení celkového objemu přenášených dat, což vedlo k výraznému zrychlení načítání.

Během vývoje jsem však přešel na distribuční větev *mainline*, která poskytovala verzi produktu 1.9. Tato verze již neobsahovala rozšíření SPDY, ale novějšího nástupce v podobě podpory protokolu HTTP 2.0. Tento protokol pracuje na stejných principech, jako SPDY, ale jeho výkon dosahuje ještě lepších parametrů. Dále je podporován ve všech moderních prohlížečích a je považován za nový standard na oblasti internetové komunikace. Tento protokol zatím stejně jako SPDY vyžaduje šifrované spojení serveru a klienta, což je velmi dobře. Bezpečnost přenosu dat přináší velkou přidanou hodnotu, avšak vyžaduje mnohem složitější konfiguraci pro správnou funkčnost. Z tohoto důvodu bylo potřeba se postarat o správné vygenerování SSL certifikátu, který není podepsán žádnou certifikační autoritou. V reálném provozu by korektní certifikát byl jednoznačně potřeba, ale pro vývojové a testovací účely jsem se rozhodl do něj neinvestovat. Dobrou alternativou by případně mohla být služba Let's encrypt od společnosti Mozilla. Ta je distributorem SSL certifikátů podepsaných certifikační autoritou, avšak na krátkou dobu. Je tedy nutné proces obnovy certifikátu automatizovat a nelze tento systém provozovat na jiných než veřejně přístupných serverech. Naopak je tato služba k dispozici zdarma.

Z důvodu použití šifrovaného připojení se serverem s nepodepsaným certifikátem se vyskytly ještě další komplikace. Moderní prohlížeče vyžadují ruční schválení varování uživatele, který takovouto stránku poprvé navštíví. Dále blokuje veškeré pokusy o připojení se k nezabezpečeným URL adresám. Toto chování je zcela správné, ale pro účely a rozsah této práce je takové chování nevhodné. Problém totiž nastal u komunikace přes WebSockets, která ve výchozím stavu probíhá nezabezpečeně. Prohlížeč tedy tato spojení automaticky zablokoval. Řešení tohoto problému je hned několik. Prvním, nejsprávnějším, je použití protokolu *wss* : // (WebSockets Secured). To s sebou nese sice mnohem lepší zabezpečení, protože veškerá komunikace tímto kanálem je také zabezpečena, ale vyžaduje poněkud složitější konfiguraci na straně serveru. Druhou možností

je dočasná deaktivace bezpečnostní politiky prohlížeče, která blokuje nezabezpečená spojení. Této možnosti jsem se také vyhnul, protože vyžadovala zásah do konfigurace prohlížeče, což by mohlo způsobit bezpečnostní problémy i u jiných navštívených stránek. Zvolil jsem proto nakonec možnost servírovat samotný HTML soubor obou klientů nezabezpečeně, což umožnilo nezabezpečené spojení i socketové komunikaci. Všechny statické soubory se však i nadále distribuují zabezpečeně a se všemi vvýhodami protokolu HTTP 2.0, což způsobuje znatelný rozdíl v rychlosti načítání především předpřipraveného data-setu.

4.8.2 Analyzátor obrazu

Druhou částí je samotný analyzátor obrazu. Ten vystavuje na portech 8010 a 8011 přípojné body pro obě klientské aplikace. Na prvním zmíneném portu očekává přijetí sekvence snímků a na druhý vystavuje výsledky analýzy. Každý vstupní snímek musí být označený unikátním identifikátorem a doplněn o konfigurační parametry, které byly popsány výše v uživatelském rozhraní producenta.

Po úspěšném přijetí snímku a doplňujících dat NodeJS obálkou, je předán ke zpracování výkonnému jádru aplikace, které je realizováno pomocí jazyka C++, jako nativní modul. Následuje převedení snímku do odstínů šedi pro detekci bodů zájmu. Ještě předtím však dojde k volitelnému preprocesingu v podobě rozostření obrazu pomocí Gaussianu. Dále dochází k detekci bodů zájmu pomocí Harrisova rohového detektoru. Výstupem této detekce je pole bodů zájmů, které obsahuje vždy idetifikátor a souřadnice pro každý bod.

Dalším, opět volitelným, krokem je shlukování pomocí metody DBScan. Na základě parametrů maximální vzdálenosti dvojice bodů a minimálního počtu bodů ve skupině jsou stanoveny shluky bodů. Odstup jednotlivých bodů od sebe určujeme pomocí euklidovské vzdálenosti, která nám popisuje vstah mezi dvojicí bodu ve 2D prostoru. Výsledné shluky však nestačí k dalším výpočtům a proto, je potřeba nalézt hranici shluků, tedy BoundingBox, kterým vymezíme oblast , kde jsme již s velkou pravděpodobností narazili na pohybující se objekt. Tento prostor pak vizualizujeme obdelníkem a dále budeme pracovat s jeho středem jako jediným bodem zájmu, nad kterým budeme provádět následné další analýzy. Díky tomuto kroku jsme schopni zpřesnit pravděpodobnost úspěsné detekce pohyblivého objektu i jeho pozici. V případě, kdy dojde ke ztrátě některého z bodů před shluknutím mezi dvěma snímky, neovlivní to tolik detekci celého objektu, protože ostatní detekované body stále udrží pozici hranice objektu relativně na svém místě. Vypořádáme se také se sirotky, které díky shlukování označíme za šum a eliminujeme je z dalšího zpracování. Stabilizujeme tak lépe detekci objektů napříč snímky, což zlepší výsledky následujících algoritmů.

Následuje proces trackování bodů mezi dvojicí snímků pomocí optického toku. K tomu je využita metoda Lucas-Kanade, která dovede nalézt spojitosti mezi množinami bodů dvojice po sobě jdoucích snímků. Body, ke kterým nebyl nalezen odpovídající obraz v předchozím snímku zahodíme, čímž opět eliminujeme další část šumu. Související dvojice bodů pak na základě konfigurace v uživatelském rozhraní zakreslíme do původního snímku a přidáme historii posledních 30

předchozích. Díky tomu bude viditelná vždy krátká trajektorie bodu, což nám umožní vizuálně lépe ověřit správnou funkčnost algoritmu. Na základě délky jednotlivých úseček můžeme také sledovat rozdíly rychlosti pohybu jednotlivých objektů.

Pro určení abnormality pohybu je potřeba na základě informací o optickém toku spočítat dodatečné parametry. Těmi jsou například rychlost, směr nebo zrychlení. Ty jsou stanoveny pro každou spojnici dvou bodů mezi sousedními snímky. Takováto spojnice je v aplikaci reprezentována instancí třídy *Motion*, která pomáhá snadněji manipulovat s doplňujícími parametry. Na základě rychlosti sestavíme zmíněný histogram pro vizualizaci distribuce různých rychlostních úrovní pohybu detekovaných bodů.

Pomocí K-means shlukovacího algoritmu dále rozhodneme o abnormalitě daného pohybu. Ještě před tím však provedeme korekci směru, pokud v obraze převládá pohyb ve svislém směru. Z informací o směru a zrychlení sestavíme vstupní pole pro klastering a na výstupu získáme rozdělení do dvou rozdílných skupin. Tu s menším množstvím bodů označíme za obnormální a tuto informaci distribuujeme zpět do třídy *Motion*, aby bylo možné vizualizovat úsečku pohybu bodu odlišnou červenou barvou. Pro lepší vizuální kontrolu sestavíme ještě 2D prostor s jednotlivými body a zakreslíme k nim středy obou shluků.

Nakonec se předá řízení programu opět do JavaScriptové části, která seskupí rozmazaný snímek v úrovních šedi, histogram, k-means klastering, původní snímek obohacený o vizualizaci pohybu a souřadnice detekovaných bodů zájmu. Tyto informace jsou pak složeny do jedné zprávy, které je odeslána na všechny připojené klienty typu konzument, kde jsou pak tyto informace zobrazeny uživatelům.

4.9 Možnosti dalšího vylepšení

Při realizaci této služby jsem narazil na mnoho míst, kde by bylo možné vylepšit nejen technické, ale také algoritmické části. Vzhledem k tomu, že je celá služba modulární, lze velmi snadno vyměnit jen dílčí část bez nutnosti větších zásahů do zbylých částí. Pokusím se tedy rozebrat alespoň základní oblasti, ve kterých by bylo možné službu vylepšit.

4.9.1 Datový přenos

Jak jsem již naznačil jednoduchým výpočtem v kapitole o producentovi videosekvence, je přenášení jednotlivých snímků videosekvence poměrně datově náročné. Při použití služby přes internet je množství přenesených dat limitováno kvalitou konektivity, což u větších rozlišení a snímkování může být limitní. Z tohoto důvodu by bylo vhodné zavést buffering na straně serveru, konzumenta i producenta. Princip této optimalizace je velmi snadný a účinný.

Vzhledem k tomu, že na internetových přípojkách dochází často ke kolísání hodnoty propustnosti, může se stát, že se po určitou chvíli setká požadovaný datový tok pro plynulé přenášení snímků s maximální propustností linky a dojde tak ke zpoždění doručení několika snímků. Tato situace může nastat jak na straně konzumenta, tak i producenta videosekvence. Vhodným ře-

šením by proto bylo zavedení mechanismu zabalení několika snímků společně s doplňujícími informacemi do jediné zprávy, která by se pak zpracovala u příjemce dávkově. Přehrávání výsledků analýzy by pak na konzumentovi bylo realizováno s určitým zpožděním, které by pomohlo pokrýt krátkodobé snížení propustnosti. Výsledné video by pak nebylo ovlivněno tolik přenosem dat a jeho přehrávání by bylo pro uživatele výrazně plynulejší. Ostatně stejnou techniku využívají všechny internetové streamovací služby v oblasti zvuku, videa či jiných dat.

4.9.2 Vyhodnocování abnormality pohybu

Aktuální způsob hodnocení pohybu je sice rychlý a vykazuje poměrně dobré výsledky. Jeho spolehlivost však má své meze, které by bylo možné posunout o něco dále. Aktuálně použitá metoda K-means je sice velmi rozšířená a vhodná pro tento typ aplikace, ale pokud chceme dosáhnout výrazně lepších výsledků, je potřeba využít poněkud odlišného principu hodnocení.

Vhodným alternativním řešením v této oblasti by bylo využití neuronové sítě. Ta by jako taková mohla téměř beze změny dalších částí nahradit metodu k-means při klasickém výpočtu. Mnohem složitější by však bylo takovou neuronovou síť naučit správnému rozpoznávání pohybu vzhledem k tomu, že máme k dispozici hned 3 různé parametry. Pojďme si alespoň ve stručnosti rozebrat možné způsoby učení.

4.9.2.1 Učení na základě směru pohybu Učení neuronové sítě dle pohybu směru objektů hodnotím jako nejméně obtížný. Vzhledem k tomu, že se objekty mohou pohybovat různými směry, měli bychom síti poskytnout alespoň 4 oddělené videosekvence, na kterých se budou objekty pohybovat nahoru, doprava, dolů a doleva. V každé videosekvenci by mělo jít vždy pouze o jediný směr, přičemž pomocí back propagation algoritmu budeme síť postupně modifikovat. Tato modifikace by měla probíhat vždy s pomocí určení jednoho směru za normální stav a ostatní směry by měly být vyhodnoceny jako abnormální.

Takovéto videosekvence není třeba nikterak explicitně ohodnocovat, obstarání takovéto čtveřice také nebude příliš problematické. Velmi důležitým krokem je však správná konfigurace parametrů preprocesingu, shlukování a optického toku. V případě i menší nepřesnosti aplikované na každý snímek videosekvence bude neuronová sít učena nekvalitními vstupními daty, což povede i k nekvalitním výsledkům její následné práce.

4.9.2.2 Učení na základě rychlosti a zrychlení Tyto dva parametry jsou mnohem obtížnější. Nelze totiž snadno získat kvalitní data-set s předem ohodnocenými rychlostmi a zrychleními v každé dvojicí snímků. Mohli bychom sice použít tento analyzátor či nějaký, již hotový, pro naměření těchto hodnot na libovolném datasetu. Přesnost neuronové sítě by však konvergovala pouze k takové přesnosti, s jakou by by scénu ohodnotil konvenční analyzátor. Kvalita výpočtu je tedy přímo závislá na kvalitě učících dat a je proto obtížné tyto data překonat.

Dalším problémem je kombinace s předchozím data-setem, který se týká směru. Dosáhnout takzvaného bezztrátového adutivního učení je velmi náročné. V případě, že bychom totiž síť

naučili rozeznávat nejdříve směr a potom rychlost a zrychlení, byla by citlivější na druhé dva parametry a méně citlivá na směr. Naopak pokud bychom nejprve naučili síť analyzovat rychlosti a zrychlení a následně teprve směr, znamenalo by to opět výhodu ve prospěch druhého parametru. Nabízí se samozřejmě možnost tyto data-sety mezi sebou opakovaně střídat tak dlouho, dokud v obou disciplínách nebude síť vykazovat dostatečně dobré výsledky.

5 Experimenty

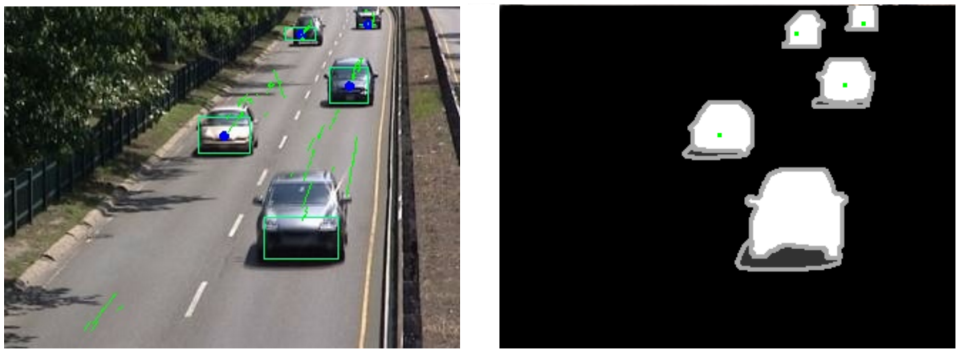
Pro ověření správnosti detekce analyzátoru jsem implementoval do aplikace konzumenta zmíněný výpočet úspěšnosti oproti již ohodnocenému data-setu. Tato hodnota mi pomohla lépe hodnotit experimentální nastavení různých parametrů a porovnávání kvality různých nastavení mezi sebou. Bylo možné sledovat závislosti mezi jednotlivými proměnnými a také rozdíly nastavení mezi různými scénami. Otestoval jsem tedy několik datasetů včetně jednoho vlastního.

5.1 Použité data-sety

Na internetu jsem nalezl několik videosekvencí, které jejich autoři připravili pro testovací účely nejen detekce pohybu. Některé obsahovaly doplňující informace se skutečným výskytem detekovaných objektů, takže bylo možné přesně změřit úspěšnost mého analyzátoru. U ostatních jsem se experimentální cestou pokusil nakonfigurovat parametry analyzátoru tak, aby vykazoval alespoň vizuálně co možná nejlepší výsledky. Pokusím se tedy alespoň stručně rozebrat některé komplikace, se kterými se analyzátor u jednotlivých videosekvencí setkával.

5.1.1 Highway

Prvním testovaným data-setem byla videosekvence s názvem "highway"ze sady CDW-2014 [3]. Jedná se o běžnou dvouproudovou komunikaci, po které jedním směrem projíždějí automobily svisle směrem ke kameře. V pravém horním rohu lze pozorovat okraj opačného jízdního pruhu. V tomto prostoru se objevuje několikrát část protijedoucího automobilu, který je po krátkou dobu také úspěšně detekován. Tento dataset jsem zpřístupnil uvnitř obou klientských aplikací pod tlačítko "Load data-set"včetně výchozí konfigurace. Lze si tedy snadno pro ukázku celou videosekvenci přehrát.



Obrázek 14: Videosekvence Highway

Tabulka 1: Konfigurace producenta pro data-set Highway

Parametr	Hodnota
Blur	30
Maximum point amount	30
Minimal point distance	8
Quality level	0.1
Block size	9
Maximal grouping distance	20
Minimal points in group	2

Tato videosekvence obsahuje také již předem ohodnocené snímky, takže lze analyzátor alespoň orientačně porovnat. V první části videosekvence bohužel masky správnosti chybí, proto nebylo možné porovnat úspěšnost detekce úplně celé sekvence. Průměrná úspěšnost detekce byla však $92,91\%$, což považuji za dobrý výsledek. Nejčastěji se objevovaly chybné detekce v horní části snímku, kdy byla dvojice jedoucích aut detekována jako jediný objekt a jejich střed byl vypočten přesně mezi ně na silnici. Dalším falešně pozitivním výskytem byl pohybující se stín na silnici v levé části záběru.

5.1.2 Pedestrians

Dalším testovaným data-setem byla videosekvence s názvem "pedestrian"také ze sady CDW-2014 [3]. Tato scéna obsahuje v horní části jízdní kolo, hydrant, schody se zábradlím a v pravé části lze pozorovat sluncem osvětlenou část parkoviště. V prostřední části je silnice s chodníky, po které se vodorovně procházejí lidé a jeden cyklista. Ve spodní části je pak už pouze trávník.

Obrázek 15: Videosekvence Pedestrian

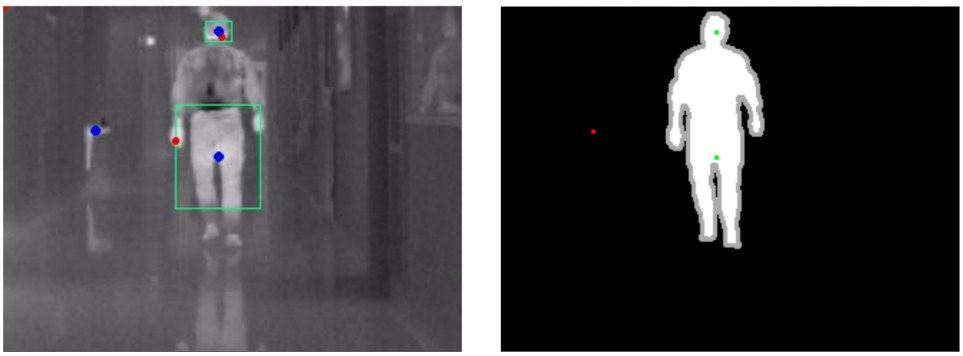
Tabulka 2: Konfigurace producenta pro data-set Pedestrian

Parametr	Hodnota
Blur	37
Maximum point amount	35
Minimal point distance	9
Quality level	0.28
Block size	10
Maximal grouping distance	29
Minimal points in group	2

Tato videosekvence vykazuje vizuálně podobné kvality analyzátoru, jako předchozí Highway, avšak v částech, kdy se ve scéně určitou dobu nepohybuje žádný chodec či cyklista, je detekován jediný bod v pravém horním rohu, kde parkuje automobil a má výraznou hranu. Tato skutečnost bohužel snižuje průměrnou úspěšnost na pouhých $62,86\%$, přestože všechny pohybující se osoby byly po dobu přehrávání videosekvence úspěšně detekovány.

5.1.3 Corridor

Videosekvence v běžně viditelném spektru jsou sice snadno dostupné, ale zajímalo mne, zda se dokáže analyzátor vypořádat i s jinými záběry. V sadě CDW-2014 [3] jsem našel tedy videosekvenci s názvem "corridor". Tato sekvence je pořízená kamerou snímající tepelné záření. Na snímcích jsou tedy místo barev zobrazeny teplotní úrovně prostředí, kde nejteplejší místa jsou nejsvětlejší.



Obrázek 16: Videosekvence Corridor

Ve scéně je vidět chodba, po které se procházejí lidé směrem od snímače a ke snímači. Po stranách jsou zřejmě kanceláře. Na chodbě se nachází také malá fontánka, ze které se procházející člověk napije. Samotný dataset je poněkud dlouhý, proto jsem vybral sadu snímků 500 - 2000, kterou jsem pomocí dostupné masky opět otestoval.

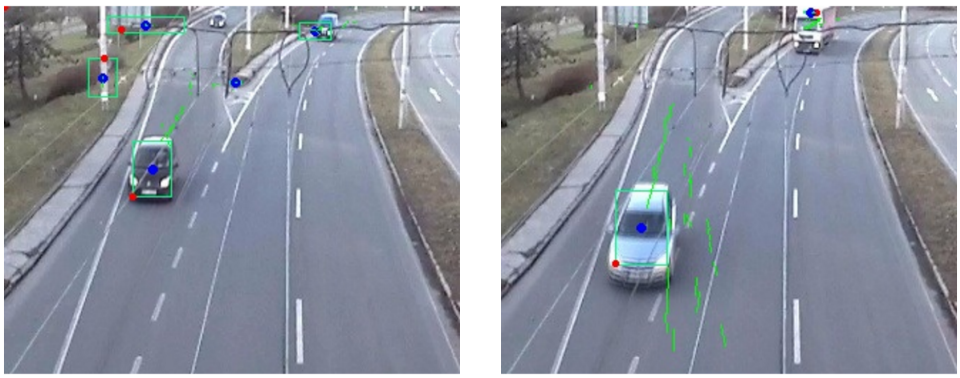
Tabulka 3: Konfigurace producenta pro data-set Corridor

Parametr	Hodnota
Blur	30
Maximum point amount	30
Minimal point distance	7
Quality level	0.13
Block size	10
Maximal grouping distance	30
Minimal points in group	2

Zmíněná fontánka tvoří ve scéně poněkud výrazný prvek, a proto je občas chybně analyzátorem označena za pohybující se osobu. Další problémy jsou viditelné při průchodu osoby chodbou směrem od snímače nebo opačně. Ve chvíli, kdy je osoba příliš blízko, je detekována jako více samostatných objektů, protože je překročen práh maximální vzdálenosti dvojice bodů od sebe ve shlukovacím algoritmu. Jinak opět vykazuje analyzátor dobré výsledky s průměrnou úspěšností $86,07\%$.

5.1.4 Vlastní data-set

Pro úplnost své práce jsem se pokusil vyrobit i vlastní data-set. K jeho pořízení jsem použil svou kameru GoPro Hero 3+ v rozlišení $1920x1080px$ s 60 snímky za sekundu. Vzhledem k tomu, že je kamera určena spíše pro pořizování sportovních záběrů, její optika vykazuje v okrajích vlastnosti rybího oka, což zkresluje výsledný obraz. Z tohoto důvodu jsem provedl středový výřez, čímž jsem se těchto deformací zbavil. Nakonec jsem ještě zmenšil rozlišení videosekvence na $320x240px$ a převedl jej na dílčí obrázky o rychlosti 30 snímků za sekundu z důvodu snížení objemu a přizpůsobení se parametrům ostatních data-setů.



Obrázek 17: Videosekvence z ulice Bohumínská

V záběru jsem zachytil ulici Bohumínskou na Slezské Ostravě z pěší lávky vedoucí z Gagarinova náměstí. Tato pozice mi umožňila pořídit stabilní záběr projíždějících automobilů směrem z městské části Muglinov na obec Vratimov. Pro zlepšení kvality záběru jsem použil uchycení kamery pomocí klasického šroubu, který jsem ukotvil k velké plastové pérové svěrce s plochými čelistmi. Ta je primárně určena k sevření dvou ploch při lepení či vrtání, a proto má velmi silnou svírací pružinu a umožňuje široké rozevření. Díky tomu jsem mohl svěrku připnout na zábradlí lávky a stabilizovat celý záběr.

Tabulka 4: Konfigurace producenta pro vlastní data-set

Parametr	Hodnota
Blur	35
Maximum point amount	34
Minimal point distance	6
Quality level	0.15
Block size	9
Maximal grouping distance	23
Minimal points in group	2

Bohužel je tato komunikace osazena trolejemi pro městskou hromadnou dopravu, které mírně komplikují práci analyzátoru. Ten však i přes tyto problémy vykazuje poměrně dobré vizuální výsledky. Nemohu však u tohoto data-setu určit jeho procentuální úspěšnost, protože bych musel nejprve pomocí jiného analyzátoru vygenerovat masky správnosti pro každý snímek. Přesto je alespoň vizuálně pozorovatelné, že si analyzátor dokáže s těmito záběry obstojně poradit.

5.1.5 Další data-sety

Podobných datasetů jsem vyzkoušel ještě mnohem více, ale podrobněji jsem zmínil pouze ty nejzajímavější. Za zmínku stojí však ještě BU-TIV (Thermal Infrared Video) Benchmark [4], který obsahuje také záběry z kamery snímající tepelné záření. Ty jsou delší a ve vyšším rozlišení, což v některých částech nevykazovalo zcela nejlepší chování analyzátoru. Naopak jsem byl velmi překvapen přesností analýzy, což bylo způsobeno především minimalizací šumu.

6 Závěr

V této práci jsem se zabýval rozborem problematiky detekce pohybu ve videosekvencích. V první části jsem uvedl a krátce rozebral dva různé přístupy k tomuto problému v kapitolách Oddělení popředí a pozadí 2.1 a Optický tok 2.2. K oběma přístupům jsem popsal několik metod řešících danou problematiku a uvedl hlavní rysy a principy. Zmínil jsem také výhody a nevýhody těchto přístupů a vhodnost jejich použití na konkrétních příkladech. Dále jsem po teoretické stránce rozebral princip fungování metody popsané v článku Abnormal Motion Detection in an Occlusive Environment [1], kterou jsem následně ve formě prototypu implementoval, abych ověřil její funkčnost.

Následně jsem se zabýval rozšířením prototypu aplikace po technické i algoritmické stránce. Navrhnul jsem a popsal architekturu softwaru, jako služby (SaaS) s ohledem na škálovatelnost tohoto řešení. Rozdělil jsem samotnou službu do několika samostatných celků a popsal proces vývoje jak po stránce organizační, tak i implementační. Pro klientské aplikace jsem použil prostředí webového prohlížeče se standardy HTML5, které komunikovaly se serverovou částí pomocí technologie WebSockets. Ty obsahují veškeré uživatelské rozhraní a ovládání služby a jsou také plně nahraditelné za alternativní implementaci pro konkrétní použití. Na serveru se mi povedlo skloubit výkon jazyka C++ se síťovou komunikací prostředí NodeJS. Nejzajímavější částí pro mne bylo nastudovat prostředí JavaScriptového jádra V8 a jeho rozhraní, pomocí kterého jsem pak integroval výkonovou část analyzátoru jako nativní modul, který jsem následně volal z interpretovaného prostředí NodeJS. Tento přístup shledávám jako velmi optimální i pro mnohé další aplikace, protože využívá toho nejlepšího z obou platforem.

Dále jsem se zabýval automatizací vývojového procesu nejen po stránce implementace, ale i infrastruktury 4.5. Pomocí nástroje Ansible jsem deklarativně popsal konfiguraci běhového prostředí a díky tomu jsem byl pak schopen pomocí několika příkazů nasadit aplikaci na vývojové i na produkční prostředí. Tento krok mi umožnil škálovat službu především horizontálně, protože konfigurace a nasazení dalších serverů s aplikací je jen otázka spuštění Ansible scriptů, které provedou veškerou práci. To mi umožňilo rozšířit analyzátor o preprocesingovou část a shlukování DBScan, což výrazně zlepšilo výsledky.

V poslední části 5 jsem již hotovou službu podrobil mnoha testům. Použil jsem dostupné data-sety k ověření kvality a připravil metriku, pomocí které jsem dovedl lépe konfigurovat vstupní parametry analyzátoru. Popsal jsem několik vybraných videosekvencí, uvedl jejich nejvhodnější parametry a naměřené výsledky. Vytvořil jsem i svůj dataset, který jsem také popsal a otestoval.

Problematika analýzy obrazu mi přišla velmi zajímavá a velmi rád bych se jí věnoval i nadále.

Literatura

[1] Panca Mudjirahardjo, Joo Kooi Tan, Hyoungseop Kim and Seiji Ishikawa, *Abnormal Motion Detection in an Occlusive Environment,* 2013.

[2] J.L. Barron, D.J. Fleet, S.S. Beauchemin, and T.A. Burkitt. *Performance of optical flow techniques.* CVPR, 92:236–242.

[3] Y. Wang, P.-M. Jodoin, F. Porikli, J. Konrad, Y. Benezeth, and P. Ishwar, CDnet 2014 *An Expanded Change Detection Benchmark Dataset* in Proc. IEEE Workshop on Change Detection (CDW-2014) at CVPR-2014, pp. 387-394. 2014

[4] Zheng Wu, Nathan Fuller, Diane Theriault, Margrit Betke, *"A Thermal Infrared Video Benchmark for Visual Analysis", in Proceeding of 10th IEEE Workshop on Perception Beyond the Visible Spectrum (PBVS)*, Columbus, Ohio, USA, 2014.

[5] Martin, Robert Cecil (2011). *The clean coder: a code of conduct for professional programmers. Upper Saddle River*, NJ: Prentice Hall. ISBN 9780137081073.

[6] Stauffer, Grimson. *Adaptive background mixture models for real-time tracking.* Cambridge, 1999.

[7] Power, Schoonees. *Understanding Background Mixture Models for Foreground Segmentation.* Auckland, New Zeland, 2002.

[8] KaewTraKulPong, Bowden. *An Improved Adaptive Background Mixture Model for Realtime Tracking with Shadow Detection.* In Proc. 2nd European Workshop on Advanced Video Based Surveillance Systems, AVBS01. Září 2001.

[9] Karmann, Brandt. *Moving object recognition using and adaptive background memory. In FRAME-RATEWorkshop*, IEEE. Elsevier Science Publishers B.V., 1990.

[10] Toyama, Krumm, Brumitt, Meyers. Wallflower: *Principles and practice of background maintenance.* In ICCV, 1999.

[11] Elgammal, Harwood, Davis. *Non-parametric Model for Background Subtraction.* In Proceedings of IEEE ICCV Frame-rate workshop, 1999.

[12] Fuentes, Velastin. *From tracking to advanced surveillance.* In Proceedings of IEEE International Confer- erence on Image Processing. Barcelona, 2003.

[13] Jean Serra *Image Analysis and Mathematical Morphology*, ISBN 0-12-637240-3, 1982

[14] Berthold K.P Horn and Brain G. Schunck *Determining Optical Flow*, Artificial Intelligence Laboratory, Cambridge, 1980

[15] Bruce D. Lucas, Takeo Kanade *An Iterative Image Registration Technique with an Application to Stereo Vision*, Computer Science Department Carnegie-Mellon University Pittsburgh, Pennsylvania 15213, 1981

[16] C. Harris and M. Stephens *A combined corner and edge detector* Proceedings of the 4th Alvey Vision Conference. s. 147–151.

A CD se zdrojovými kódy

Na tomto CD jsou k dispozici zdrojové kódy pro spuštění lokálního vývojového prostředí služby. V souboru *README.md* je k dispozici stručný návod na instalaci a spuštění tohoto prostředí. Dále příloha obsahuje zdrojové kódy všech částí aplikace spolu se skripty pro sestavení. Také se zde nachází vlastní dataset.